Natural Language Processing (NLP)

Word embeddings, 1D CNN, LSTM and Transformers

Zaur Gouliev | 12-01-2025

PhD Student, University College Dublin
zaur.gouliev@ucdconnect.ie

Acknowledgements

ZAUR GOULIEV

PhD Student, University College Dublin

---

http://zaurgouliev.com

Third party sources: cited per item in slides.

00 Introduction

- Words
- Challenges
- One-Hot-Encoding

01 Data Processing / Context

- Tokenising
- Context
- Word Embeddings

02 LSTMs

- DANN and CNN sentiment
- LSTMs sentiment
- LSTMs spam
- LSTMs for other taksks

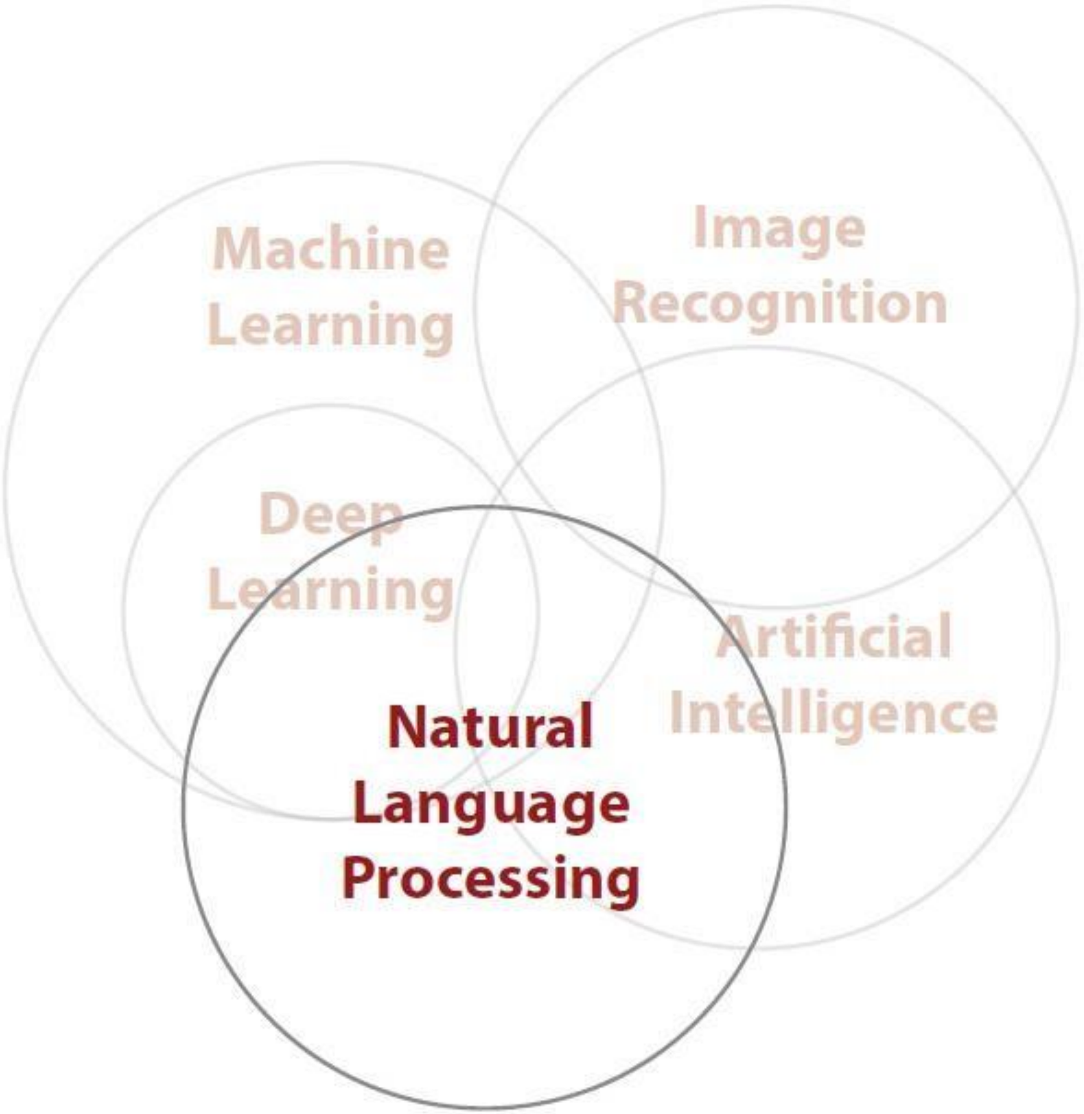
03 Transformers

- Introduction to LLMS
- Transformers
- BERT
- GPT

0) Introduction

"Action speaks louder than words but not nearly as often."

Mark Twain



NLP

"Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyse large amounts of natural language data."

Wikipedia

“Look before you leap.”
Charlotte Bronte

&

What’s the story?

Working with words

Firstly, words are not numeric

Most topologies require numeric inputs

Context matters (a lot)

Order matters (a lot)

How might we change the following to work with a NN:

“Look before you leap.”

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No

	outlook	temp	humidity	wind	play
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0

Challenges

We can encode the words into numeric values, we have already seen examples in the earlier labs using label encoding:

```
data = pd.read_csv("weather.csv")

print("\n\nRaw Data\n\n",data)


data = data.apply(preprocessing.LabelEncoder().fit_transform)
data.hist()
print(data.describe())
```

In this example:

Sunny	**->**	**2**
Overcast	**->**	**0**
Rain	**->**	**1**

Will this be sufficient?

“Look before you leap.”
[1, 0, 3, 2]

“Leap before you look”
[2, 0, 3, 1]

Challenges

Using word encoding what might this look like?

How could we feed this into a neural network?

Taking the two word/input vectors in the example, there are some challenges:

- The first input contains a 1 and a 2, these are different values, and don’t map the meaning as for instance one it is look and two it is leap.
- This challenge is similar to the image classification problems and the loss of special awareness.

Next, let’s look at techniques that may help.

Look = [1, 0, 0, 0]

Before = [0, 1, 0, 0]

You = [0, 0, 1, 0]

Leap = [0, 0, 0, 1]

One-Hot-Encoding

Another form of storage is called "one-hot encoding" and is the most common format for encoding binary/categorical data.

The "binary" presence or absence of an input datapoint amongst a vocabulary of possible input datapoints. There are other forms of encoding such as using the sum of each word occurace in a sentence, but one-hot–encoding typically outperforms this.

So, if our vocabulary was only 4 words, our one-hot encoding might look like the following.

Look = [1, 0, 0, 0]
Before = [0, 1, 0, 0]
You = [0, 0, 1, 0]
Leap = [0, 0, 0, 1]

“Look before you leap”
[1, 1, 1, 1]

“Leap before you look”
[1, 1, 1, 1]

One-Hot-Encoding

Challenges:

When we encode the two sentences as one hot encoded vectors, based on the language input, we get the same vectors.

This has significant consequences for “context” on whatever form a model can learn context, as in this case, it has no chance, if very different sentences result in the same hot encoded vector.

1) Data Processing / Context

Tokenizing and Word Embeddings

Definition to Tokenize the inputs

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# definition to tokenize the inputs
def tokenizer_sequences(num_words, X):

    # when calling the texts_to_sequences method, only the top num_
    # the Tokenizer stores everything in the word_index during fit_

    tokenizer = Tokenizer(num_words=num_words)

    # From doc: By default, all punctuation is removed, turning the
    tokenizer.fit_on_texts(X)
    sequences = tokenizer.texts_to_sequences(X)

    return tokenizer, sequences
```

Tokenizing

We need to work with text, by applying tokens (numeric values to the text).

There are many options here, for example:

- FCFS
- Based on frequency of occurrence
- Also for Word Embeddings (up next) we also need to select num_words in the vocab to use (hyperparameter).
- Returns Tokenizer and Sequences

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

Only use the X column (named email) and tokenize the inputs

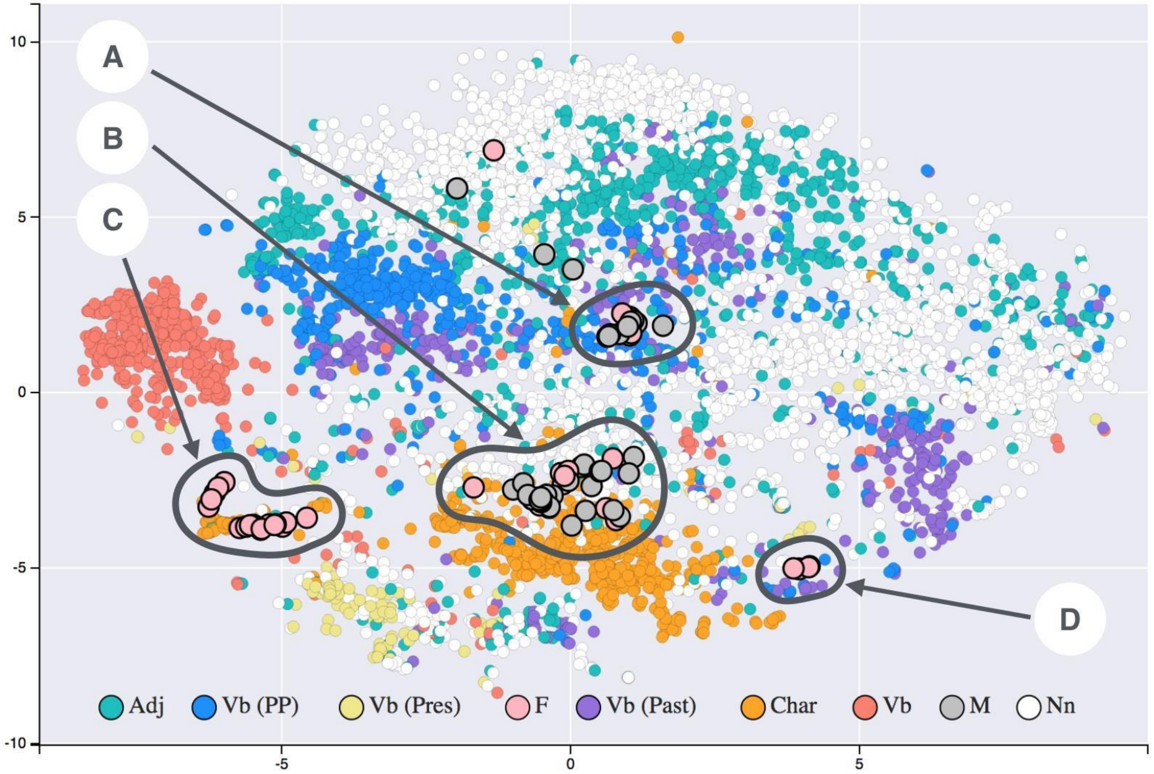
```
tokenizer, sequences = tokenizer_sequences(max_words, df["email"].copy())

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
print("Using top: ", max_words,"tokens.")

print("Padding/shortining all emails to ", maxlen, "words.")
X = pad_sequences(sequences, maxlen=maxlen)
y = df["label"].values

np.random.seed(1)

# randomize the data set - numpy arrays
randomize = np.arange(len(X))
np.random.shuffle(randomize)
X = X[randomize]
y = y[randomize]

print('Shape of data :', X.shape)
print('Shape of label:', y.shape)

print(X[:10])
```

```
Found 33672 unique tokens.
Using top:  10000 tokens.
Padding/shortining all emails to  300 words.
Shape of data : (2999, 300)
Shape of label: (2999,)
[[   0    0    0 ...    1    1    9]
 [ 369    1    1 ... 1024 1761 1134]
 [   0    0    0 ... 3012 1168 6746]
 ...
 [   0    0    0 ...  470  491    9]
 [   0    0    0 ... 2140 2465  254]
 [3160    4  471 ...  200    9    9]]
```

Tokenizing

- Tokenizer (top 10,000 tokens)
- Padding (300 words)
- Sequences 0 is non word (reserved)

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer



Context using word embeddings

Let’s start with some word embedding history and concepts.

- What are embeddings
    1D and 2D – 32D+
- Word2vec
- Bag of words and skip-o-grams

- Keras embeddings

https://commons.wikimedia.org/wiki/File:T-SNE_visualisation_of_word_embeddings_generated_using_19th_century_literature.png

Look before you leap, John said to Jane

1 17 0 2 19 6 0 81

Context using word embeddings

Let's start with some word embedding history and concepts.

We will begin with the idea of a 1D embedding, what might it look like?

Left is the input in tokenized and padded

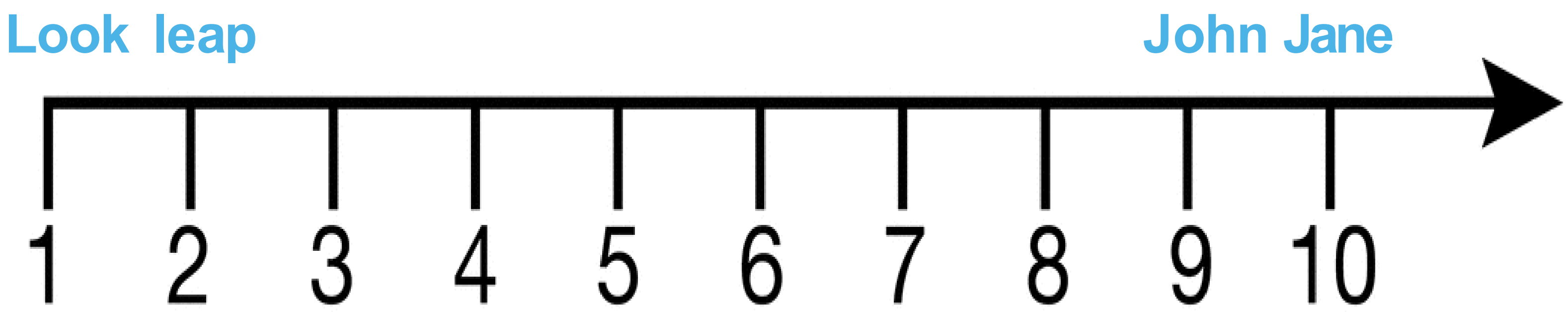
Look before you leap, John said to Jane

1 2 9 10

Context using word embeddings

Let's start with some word embedding history and concepts.

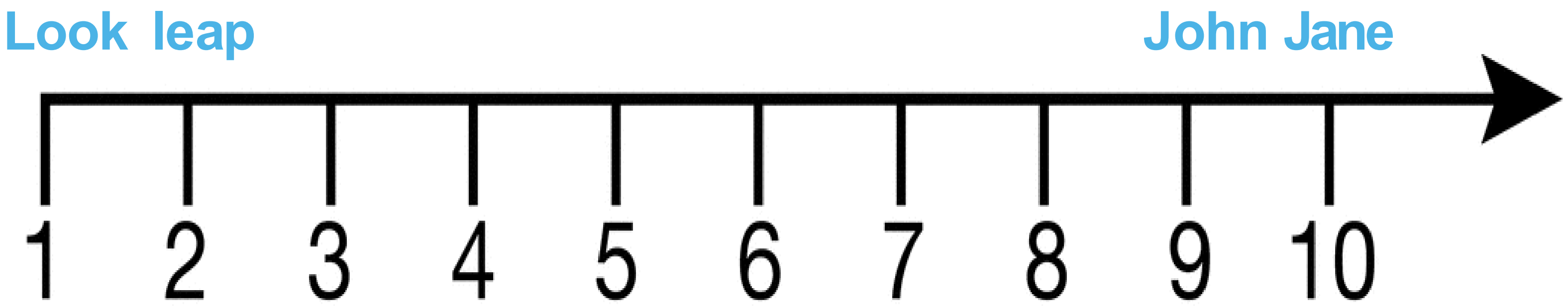
Above is using a coding with a 1D vector
(I have ignored some values)



Look before you leap, John said to Jane

1 2 9 10

Context using word embeddings



Similarity

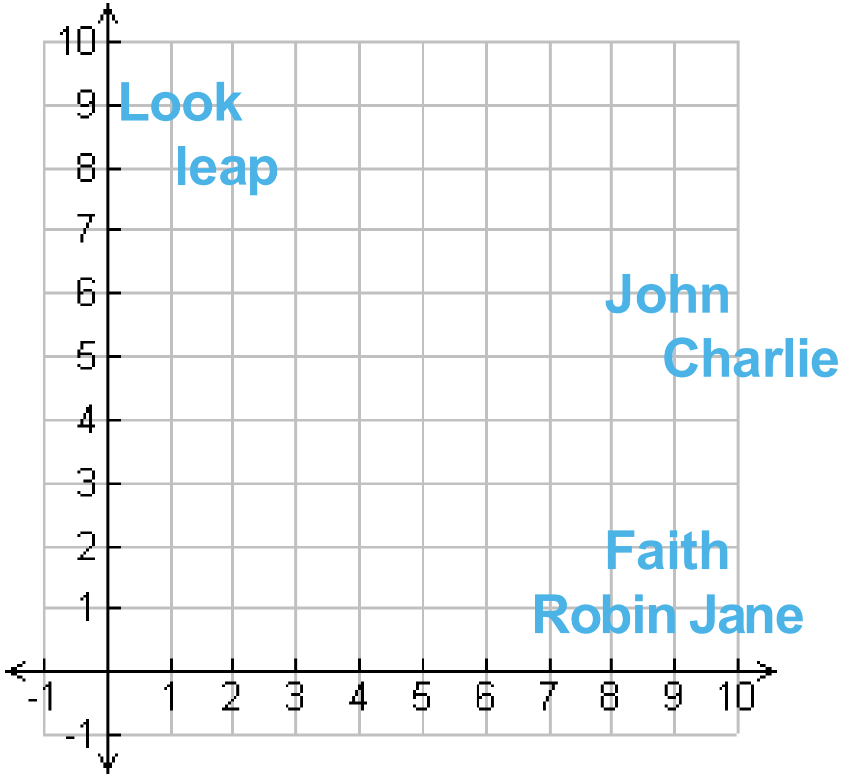
Look leap 2 - 1 = 1

John Jane 10 – 9 = 1

John leap 9 - 1 = 8

Look before you leap, John said to Jane

[1,9] [2, 8][9, 6] [10, 1]



Context using word embeddings

When we move to 2D, we might have more associations.

We get more associations/similarities, such as:

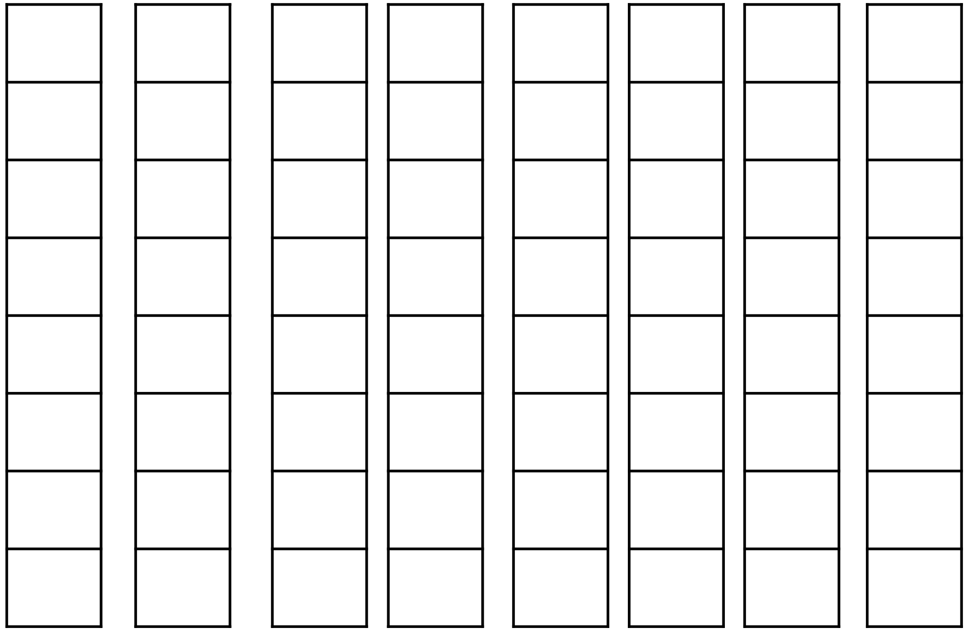
Using:

Distance Formula

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

What similarities might exist?

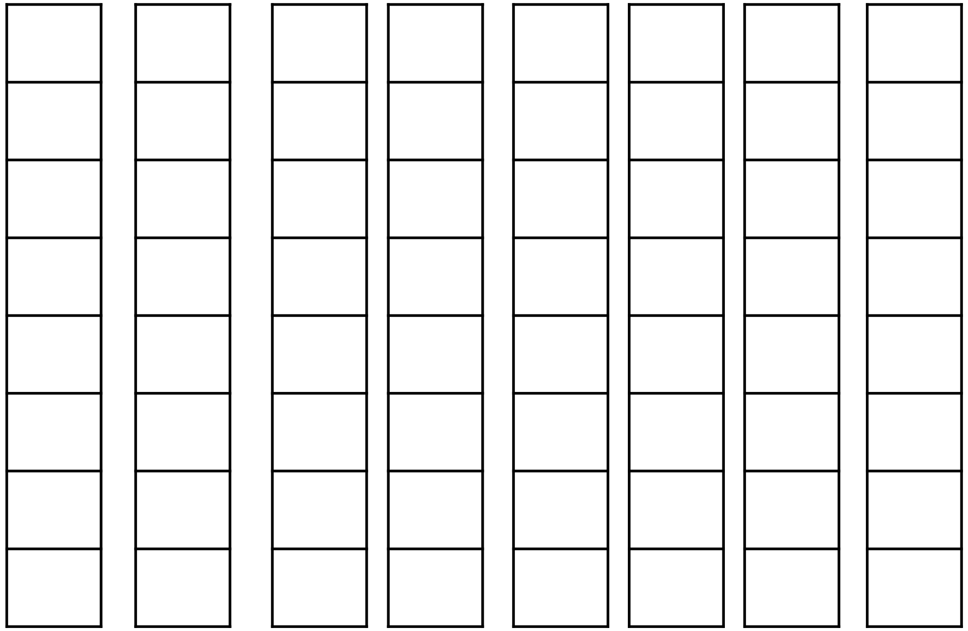
Look before you leap, John said to Jane



Context using word embeddings

What they look like (8 Dimensions)

Look before you leap, John said to Jane



Context using word embeddings

What they look like (8 Dimensions)

Dimensions

Word vectors

dog	-0.4	0.37	0.02	-0.34
cat	-0.15	-0.02	-0.23	-0.23
lion	0.19	-0.4	0.35	-0.48
tiger	-0.08	0.31	0.56	0.07
elephant	-0.04	-0.09	0.11	-0.06
cheetah	0.27	-0.28	-0.2	-0.43
monkey	-0.02	-0.67	-0.21	-0.48
rabbit	-0.04	-0.3	-0.18	-0.47
mouse	0.09	-0.46	-0.35	-0.24
rat	0.21	-0.48	-0.56	-0.37

animal
domesticated
pet
fluffy

OHE → Word2Vec → Word Embedding

Word2Vec

One hot encoded values are fed in, and a word embedding is created

There are generally two forms for this:

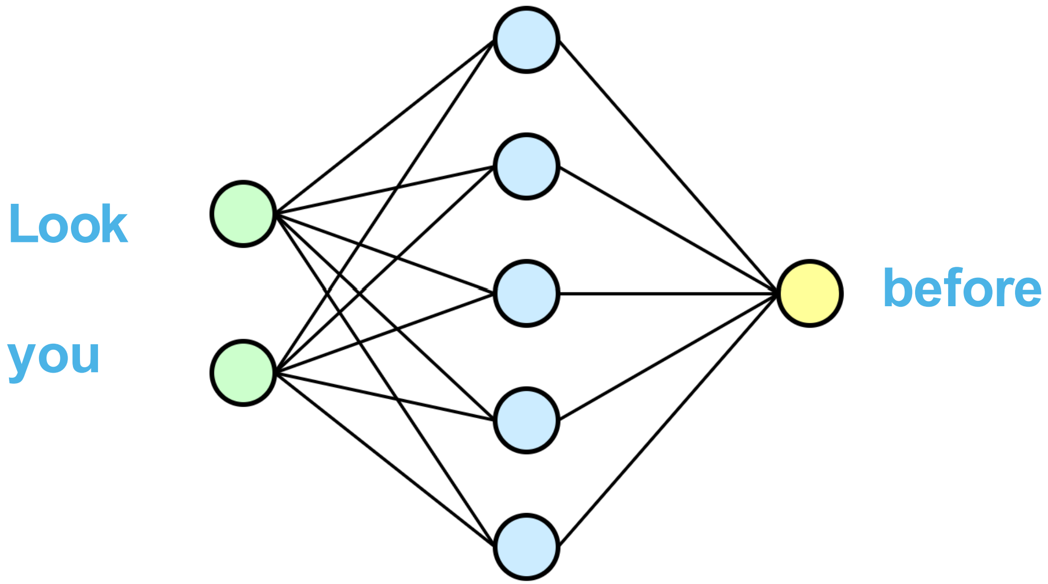
- Continuous Bag of Words
- Skip gram

These are often called custom embeddings

Look before you leap, John said to Jane

OHE → Word2Vec → Word Embedding

Look before you leap, John said to Jane



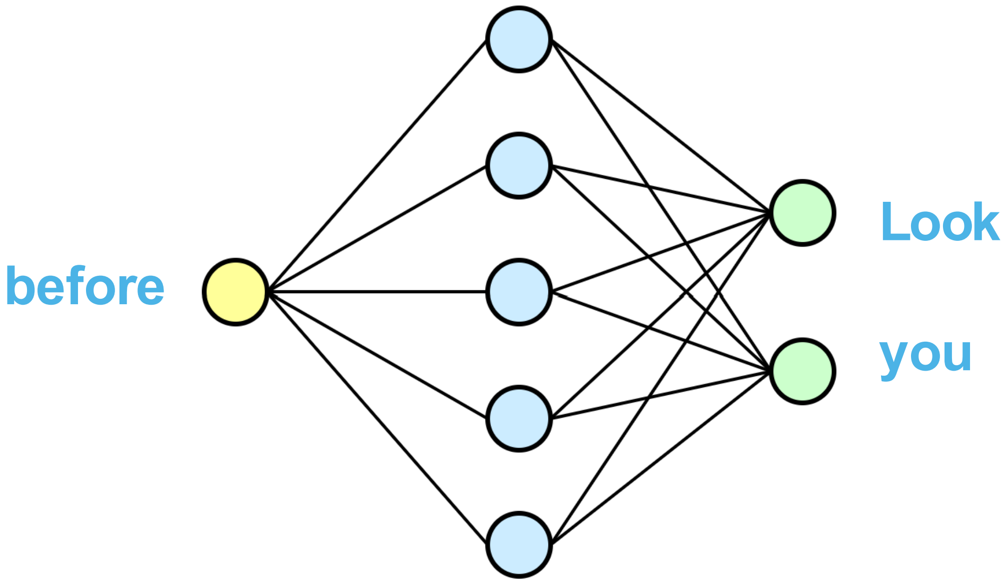
Word2Vec

One hot encoded values are fed in, and a word embedding is created

There are generally two forms for this:

- **Continuous Bag of Words**
- Skip gram

Look before you leap, John said to Jane



Word2Vec

One hot encoded values are fed in, and a word embedding is created

There are generally two forms for this:

- Continuous Bag of Words
- **Skip gram**

```python
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=5000)
```

```python
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
```

```python
# create the model
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
```

Keras

This is trained at run time, this the similarities/weight updated.

The code left, has:

- a max number of words as 5000
- Input size of 500 (padded)
- Output size or 32 vectors

https://keras.io/api/layers/core_layers/embedding/

Note: can also seed embedding (if training data s small) with a Word2Vec model to provide initial context, similar to transfer learning.

https://commons.wikimedia.org/wiki/File:T-SNE_visualisation_of_word_embeddings_generated_using_19th_century_literature.png

2) CNNs & LSTMs

Prior to transformers

```python
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=5000)
```

```python
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
```

```python
# create the model
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

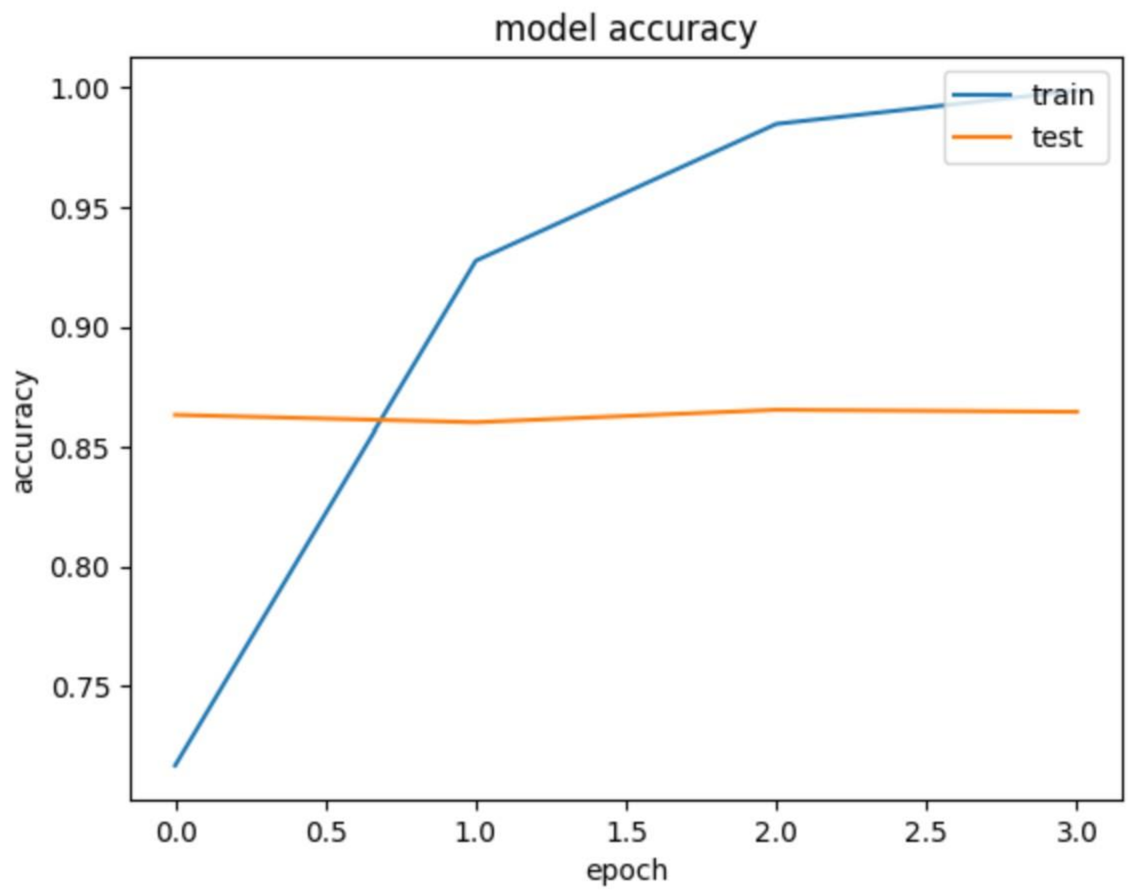
DANN

We can use vanillia ANNs once we feed in the word embedding as the input

This significantly improved NLP models in the past.

Note where flattening is, why?

Accuracy: 86.43%



DANN

We can use vanillia ANNs once we feed in the word embedding as the input

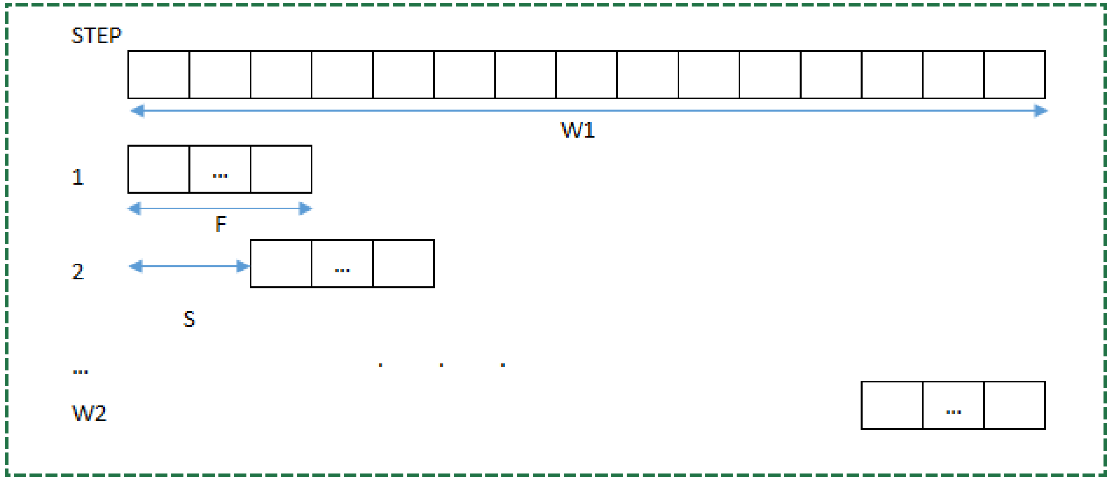
This significantly improved NLP models in the past.

Note: it is a very small network

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=5000)
```

```
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
```

```
# create the model
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```



1D CNN

We can also use Convolutional Neural Networks for text.

This allows for a sliding window.

We can also feed in the word embedding

1D CNN

```python
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation="relu"))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

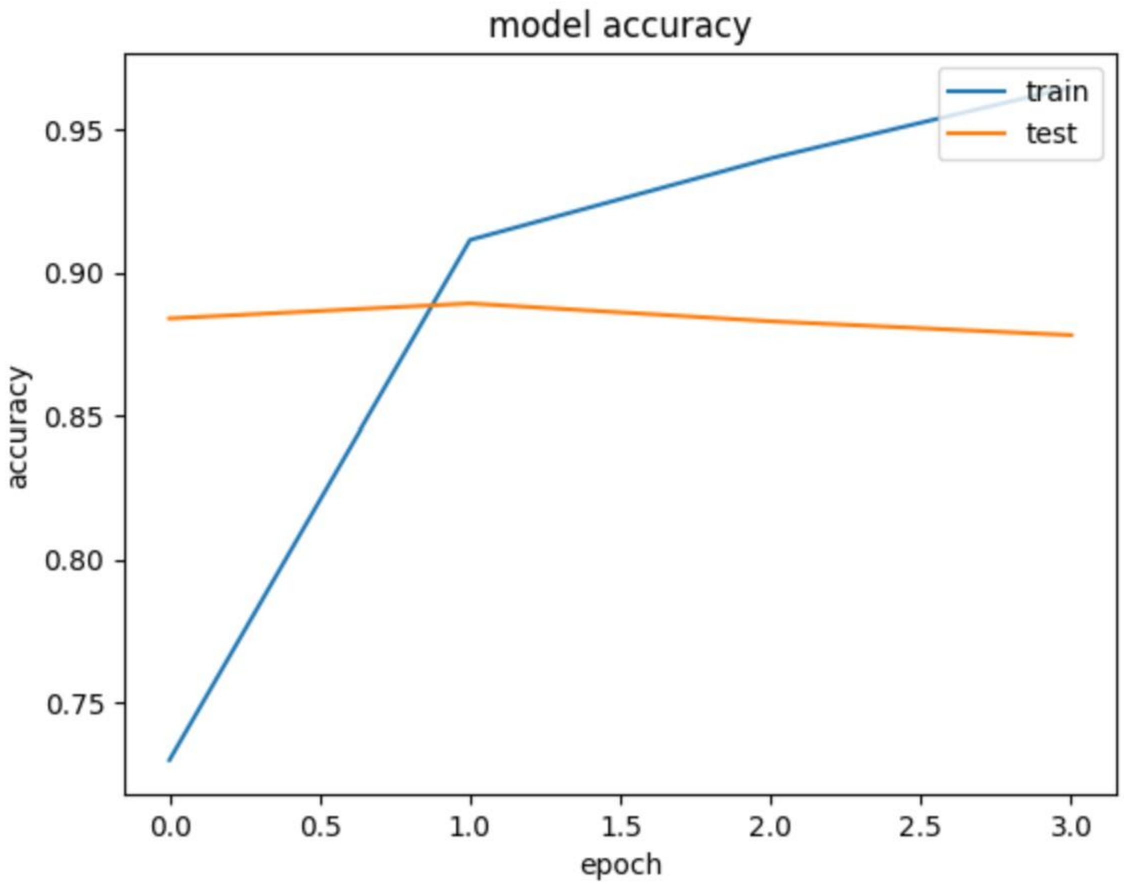
We can also use Convolutional Neural Networks for text.

This allows for a sliding window.

We can also feed in the word embedding.

Note where flattening is, why?

Accuracy: 87.82%



1D CNN

We can also use Convolutional Neural Networks for text.

This allows for a sliding window.

We can also feed in the word embedding

```
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation="relu"))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

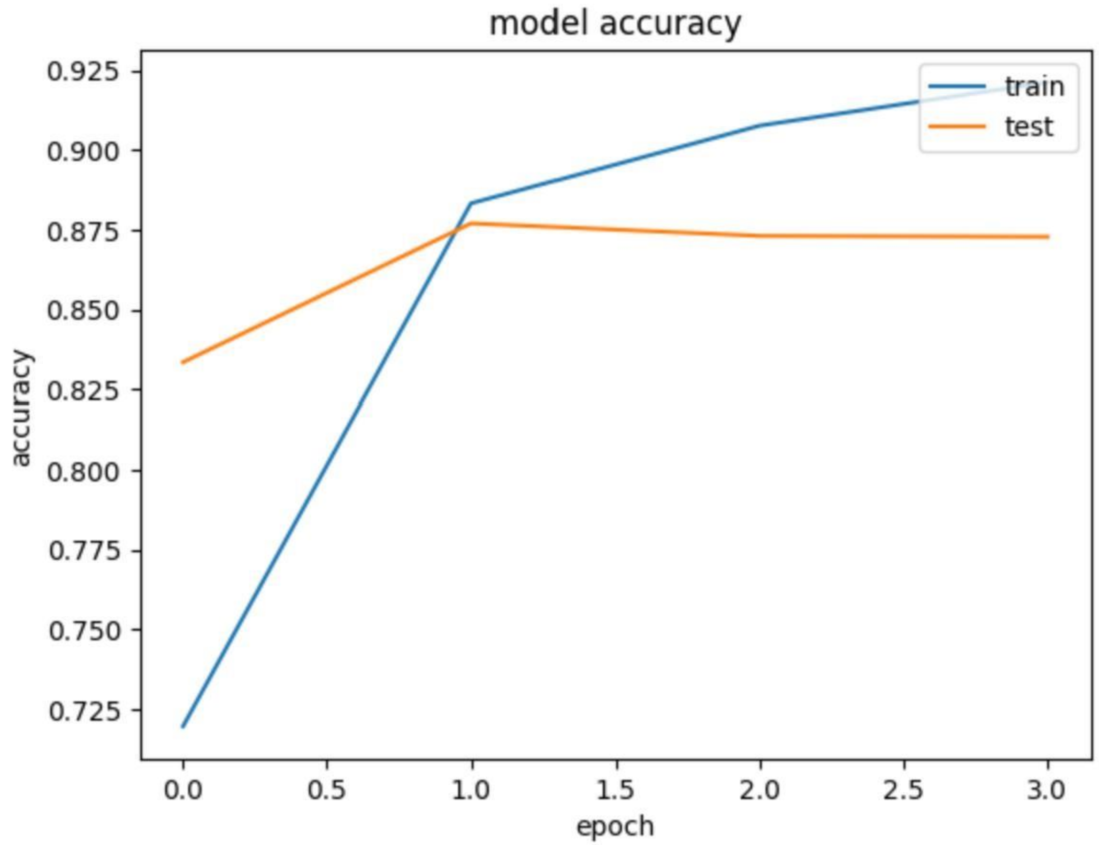
```
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(LSTM(128))
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

LSTMs

LSTMs also work well, in tandem with word embeddings.

Why is there no flattening?

```
Accuracy: 87.28%
```



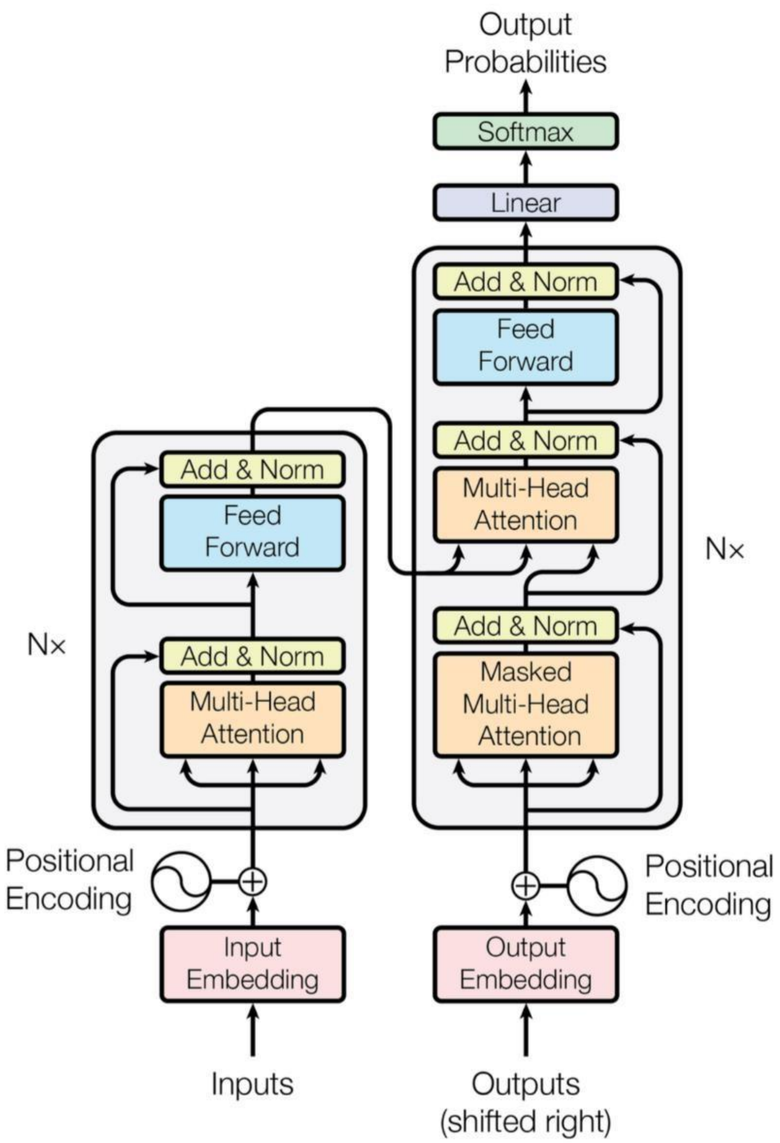
LSTMs

LSTMs also work well, in tandem with word embeddings.

```
model = Sequential()
model.add(Embedding(5000, 32, input_length=500))
model.add(LSTM(128))
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

3) Transformers

Positional Encoding, Attention, Multi-Head Attention, Bert and GPT



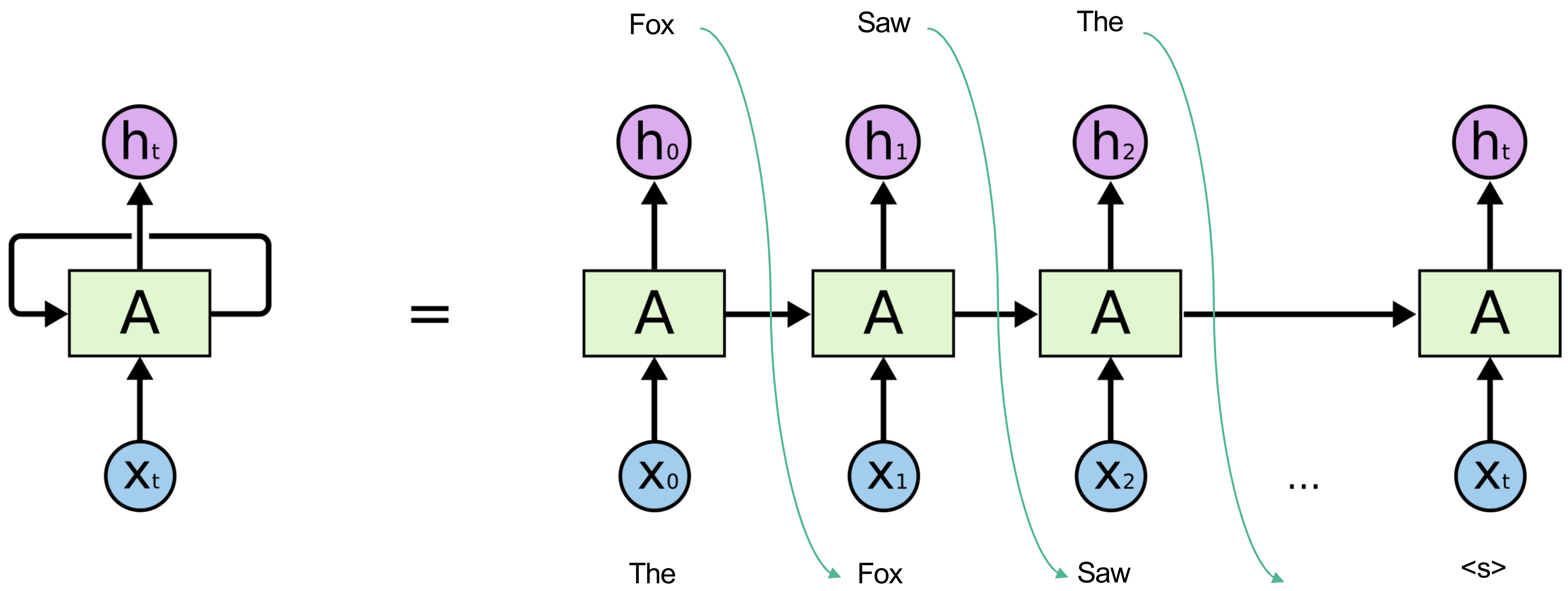
Transformers

Overview:

- The end of RNNs and LSTMs?
- Introduction to transformers
- Attention is all you need
- Bert and ChatGPT

Transformers

Recap on LSTMs (RNNs)

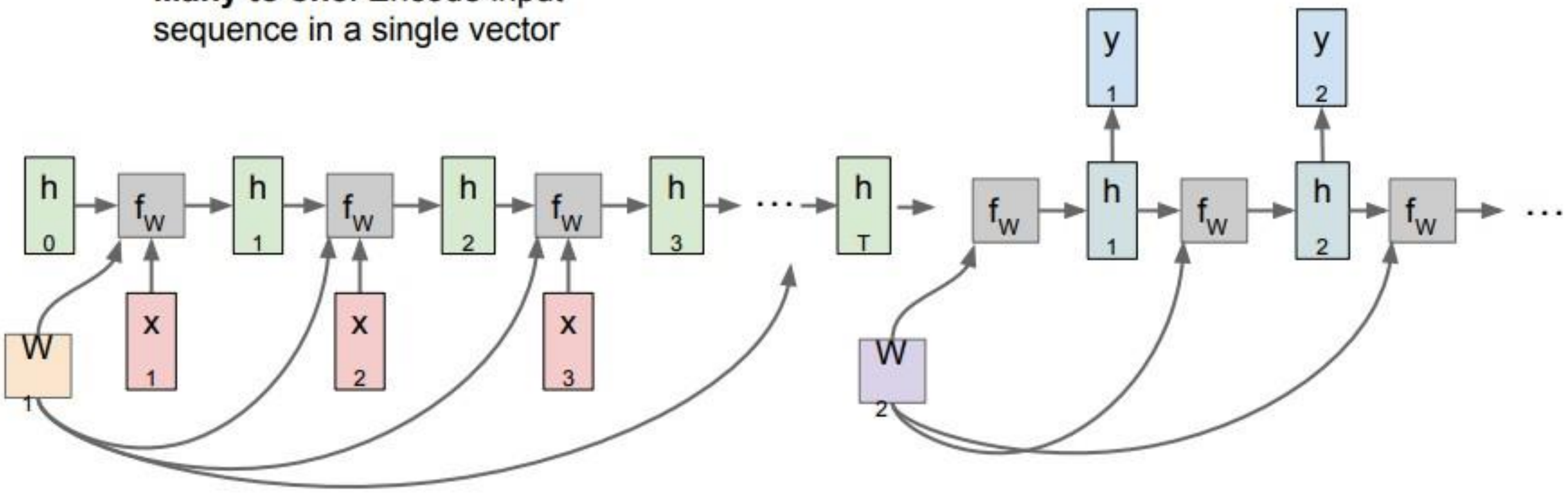


Transformers

Recap – Encoding and Decoding

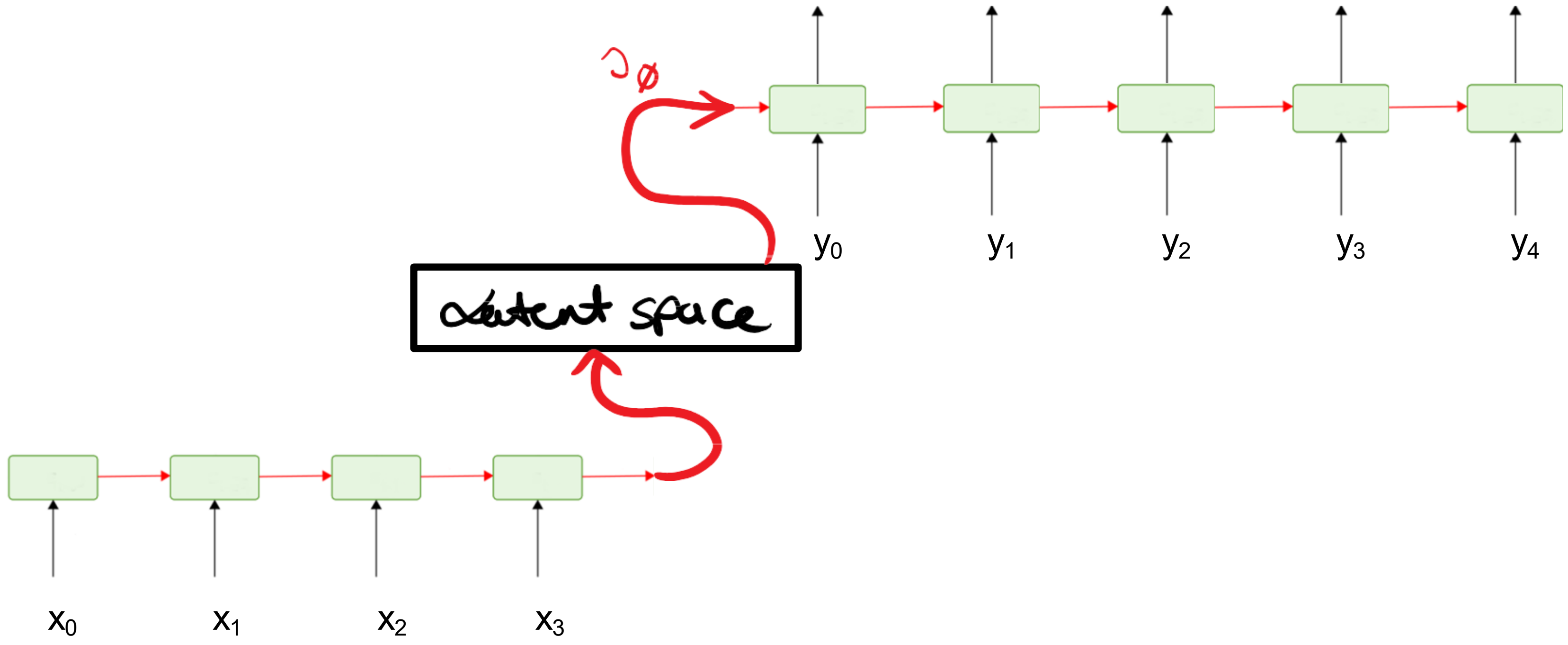
Many to one: Encode input sequence in a single vector

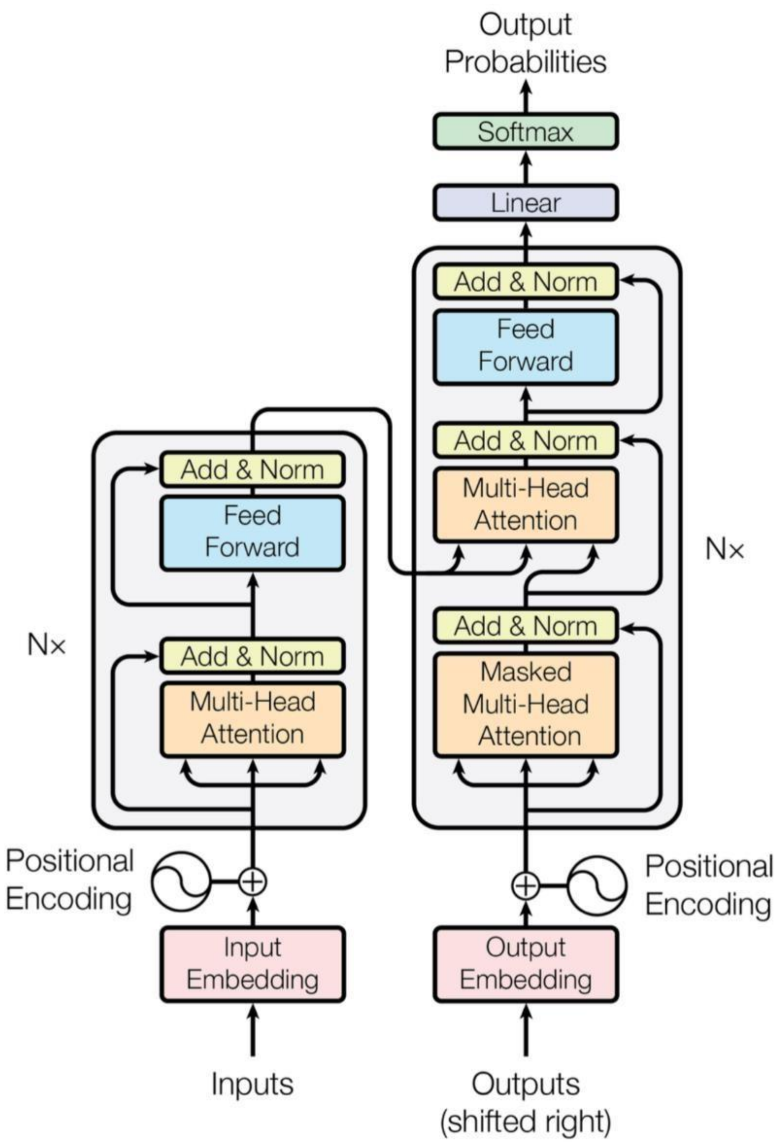
One to many: Produce output sequence from single input vector



Transformers

Recap: Encoding and Decoding





Transformers (2017)

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

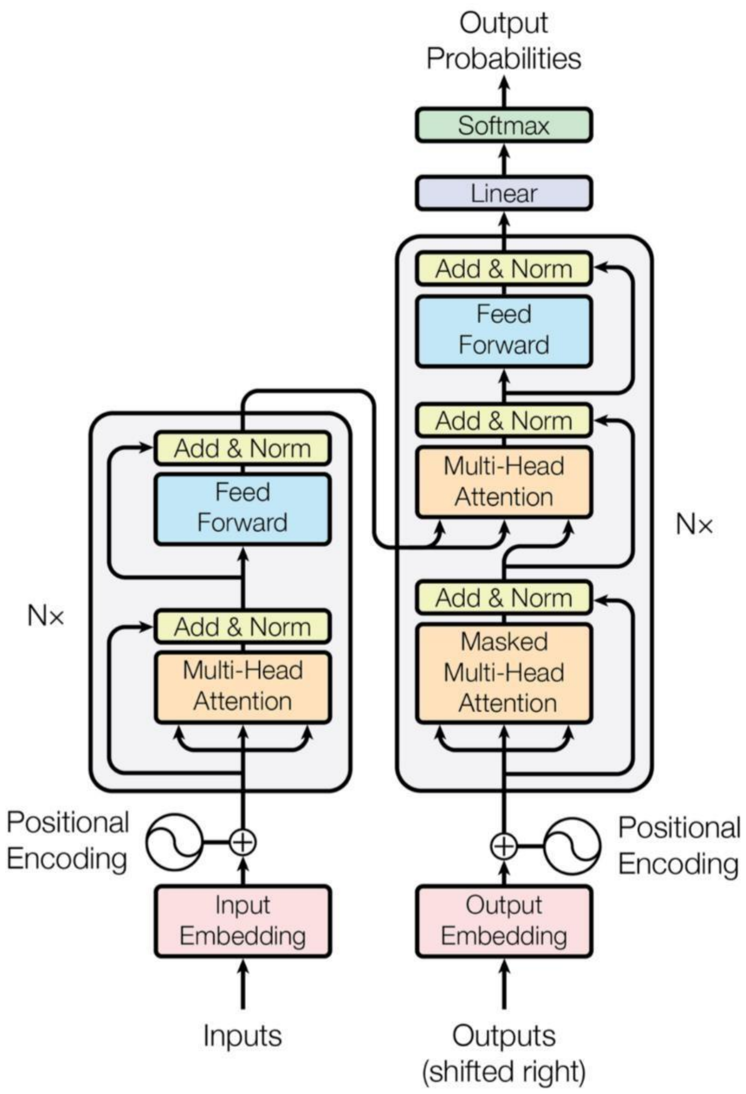
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

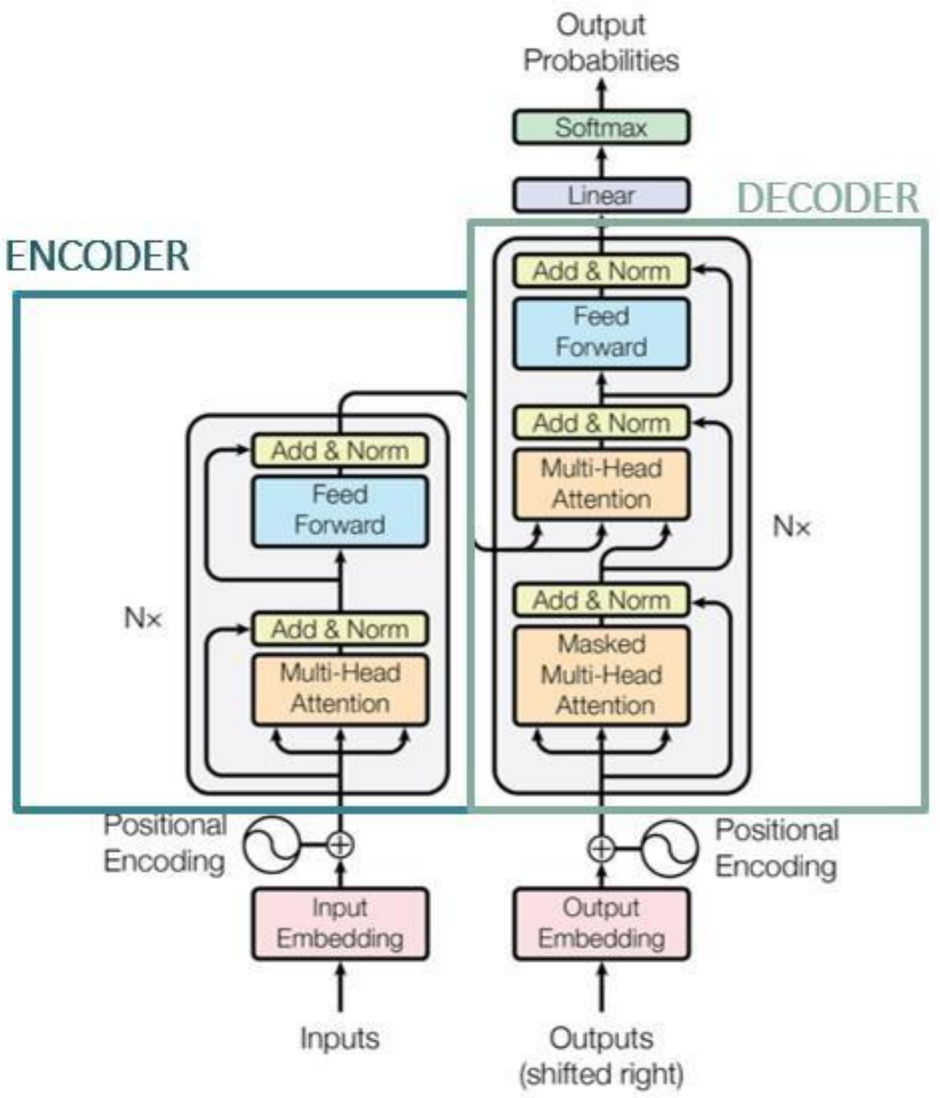
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions



Transformers

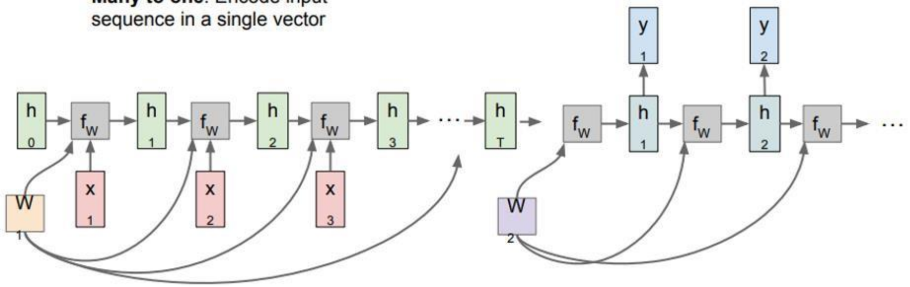
Why are they more popular?

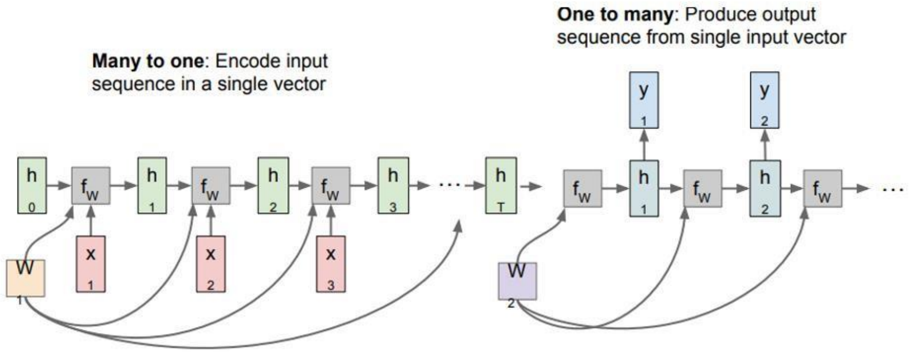
- Uses attention to remember
- No recurrence
  (ordered input relying on previous tokens)
- Faster to train
- Can be parallelised

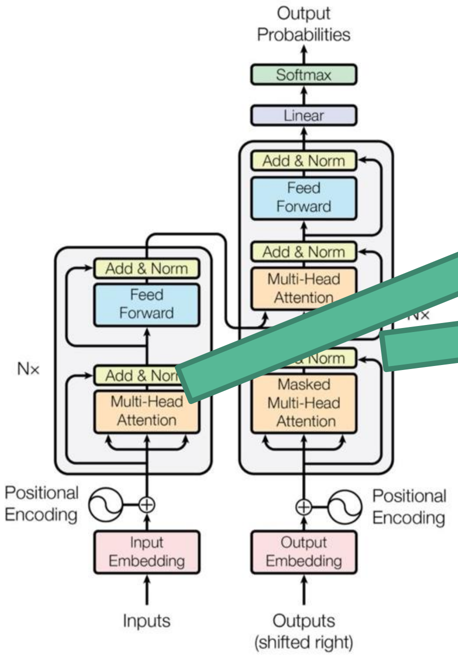


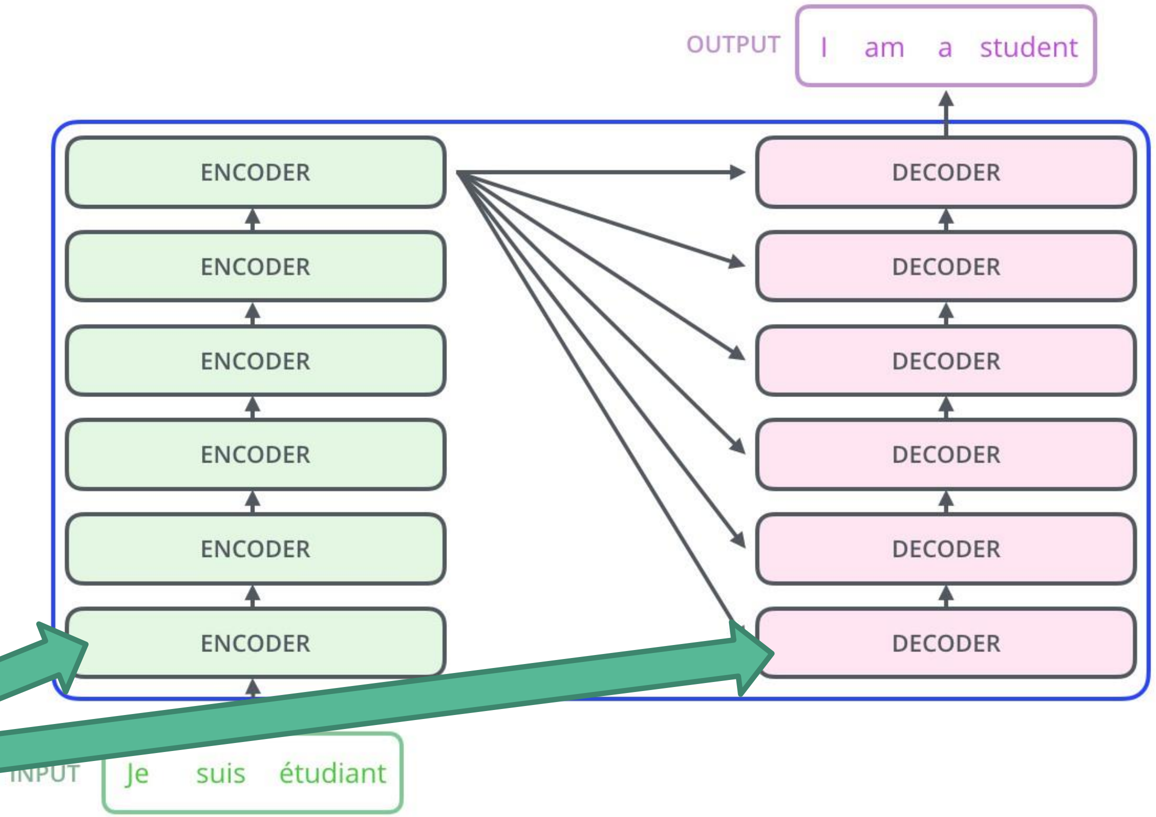
Transformers

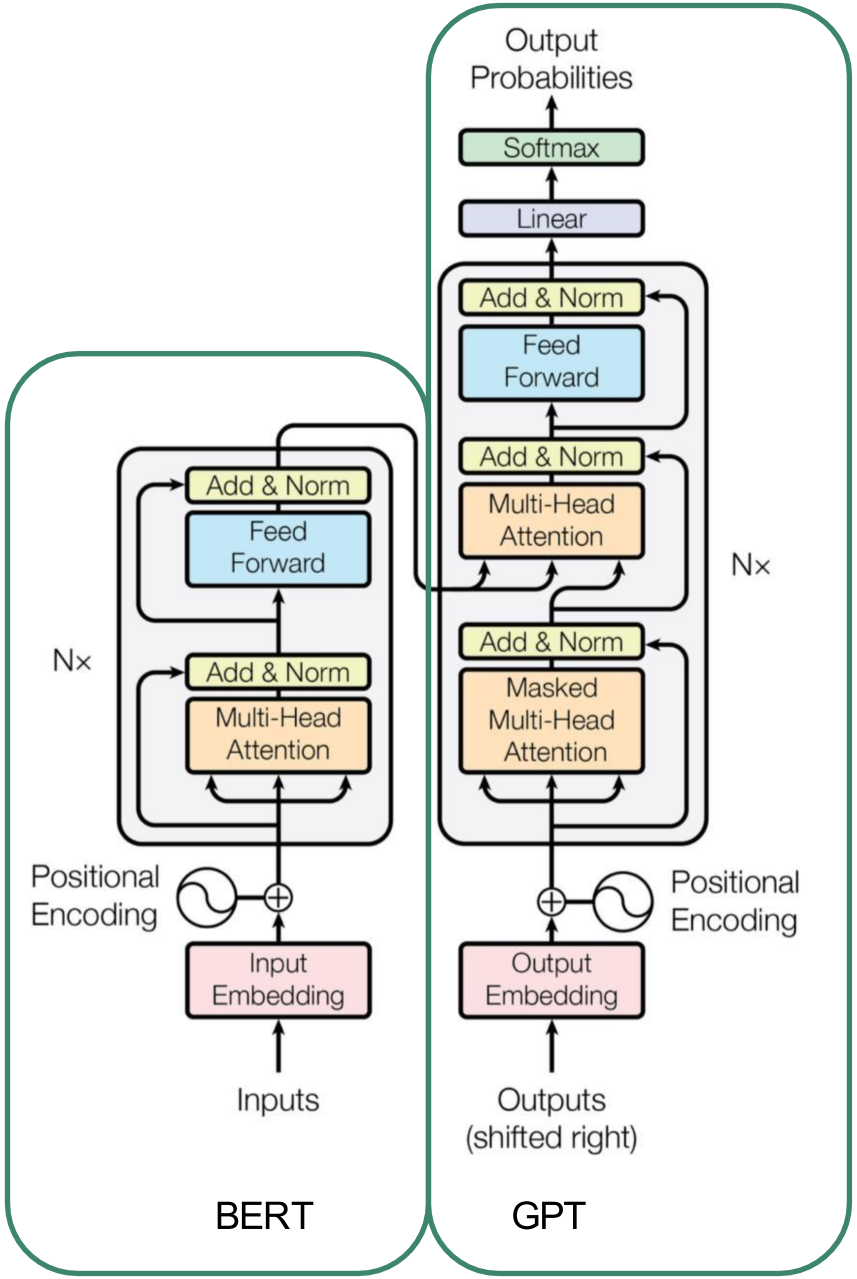
A transformer consists of two components an encoder and a decoder.




Many to one: Encode input sequence in a single vector
One to many: Produce output sequence from single input vector
h_0 f_W h_1 f_W h_2 f_W h_3 … h_T f_W h_1 f_W h_2 f_W …
W_1 x_1 x_2 x_3 W_2 y_1 y_2


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

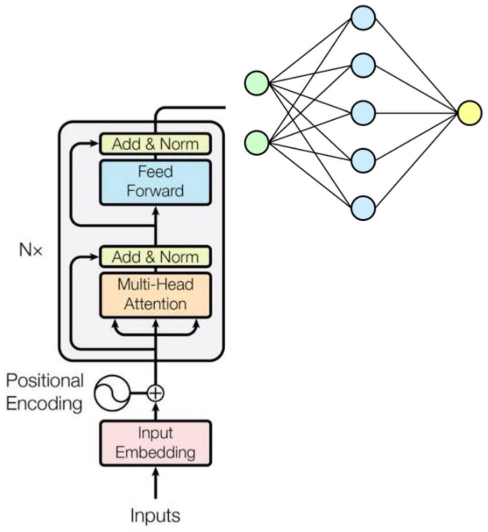

OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

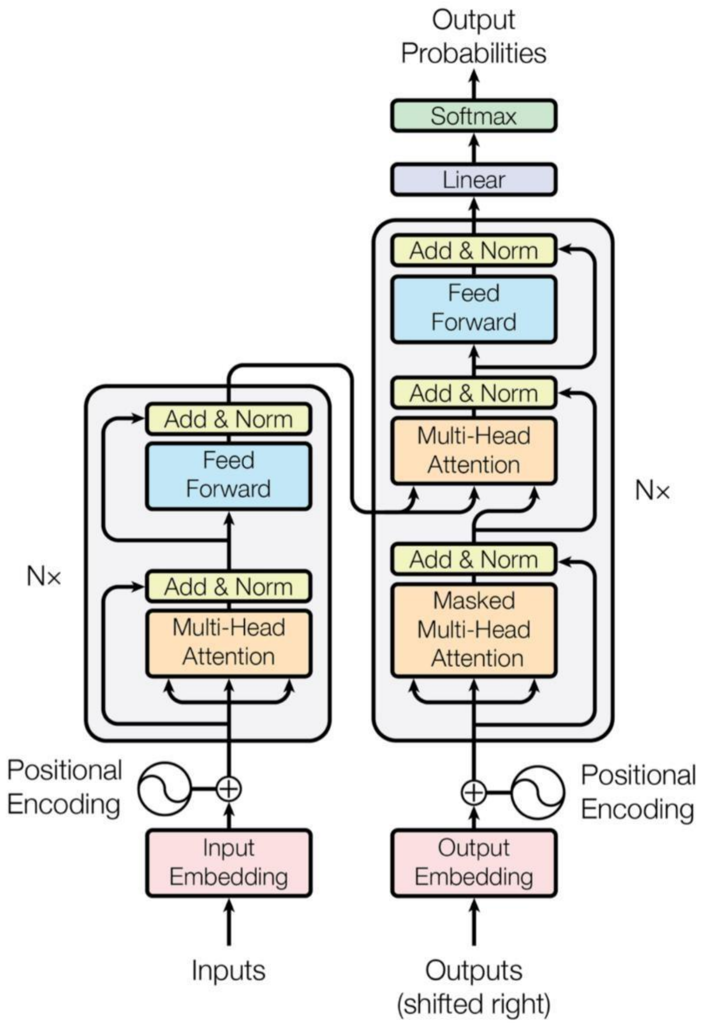


Transformers

These two components have been implemented in models that you may have heard of.

We simply add a dense neural network at the end points of one or the other component.





Transformers

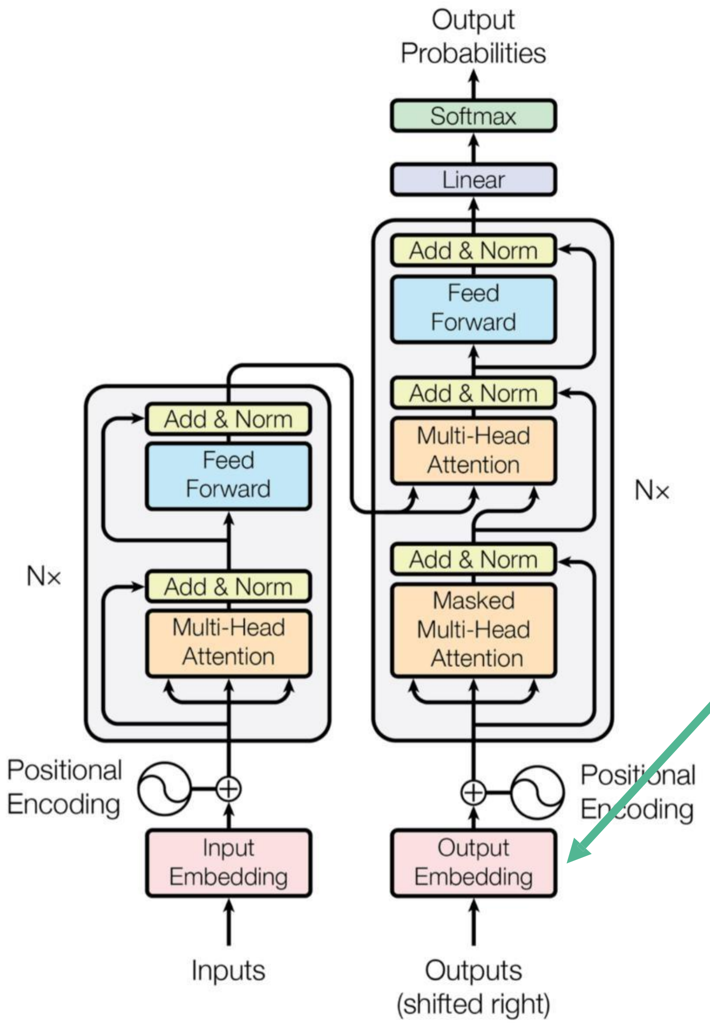
The original model: Attention

Encoder:

- One multi-head attention
- One feed forward ANN

Decoder:

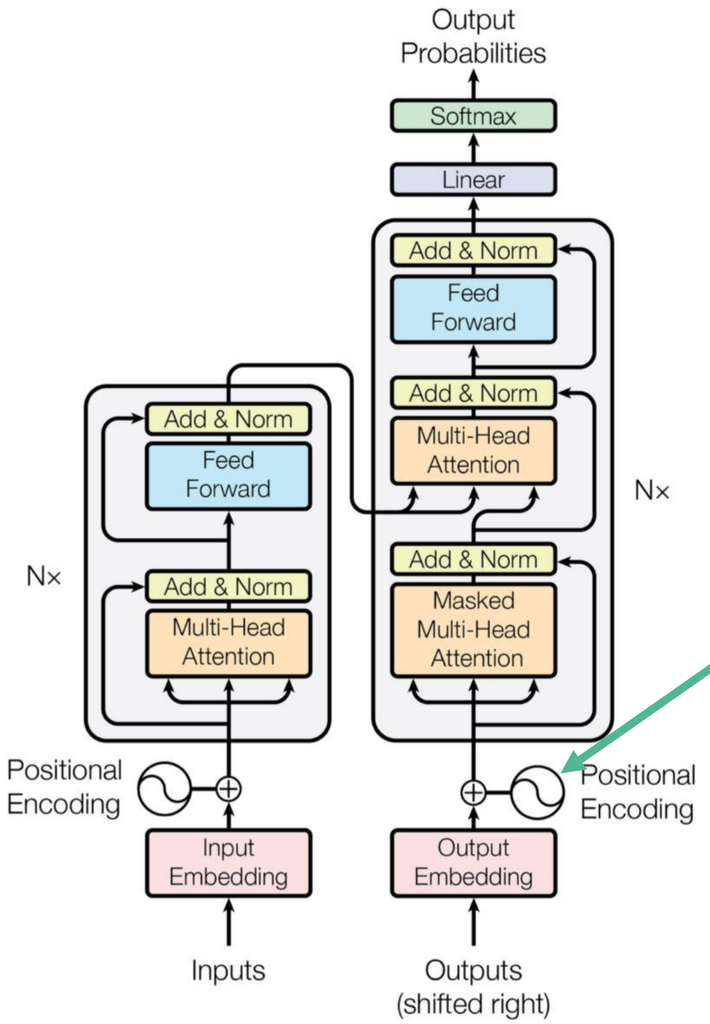
- Two multi-head attention
- One feed forward ANN



Transformers

Transformers consist of:

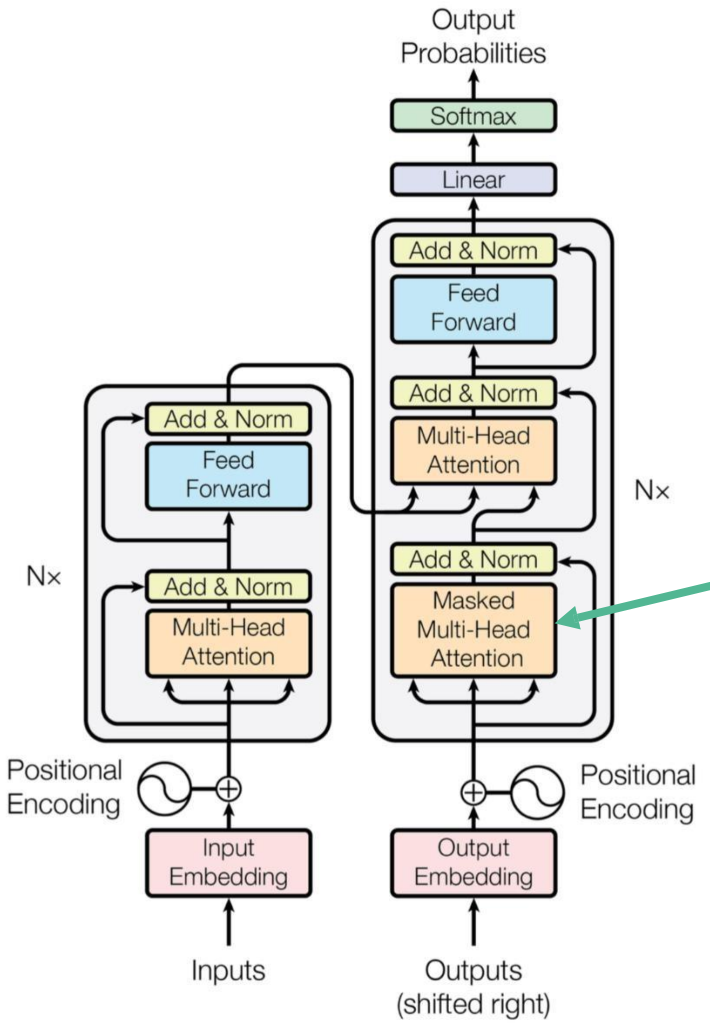
- Word Embedding
- Positional Encoding
- Attention



Transformers

Transformers consist of:

- Word Embedding
- Positional Encoding
- Attention



Transformers

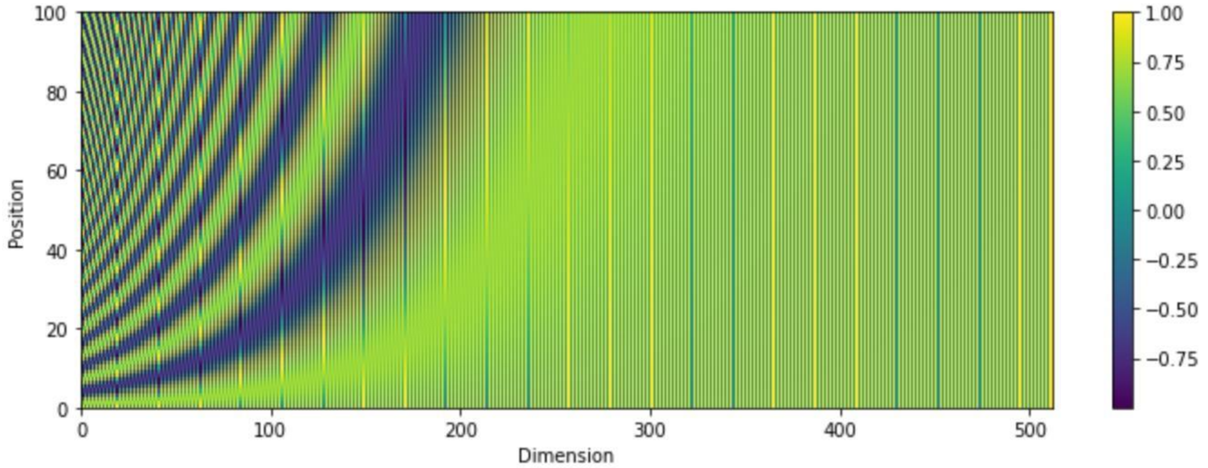
Transformers consist of:

- Word Embedding
- Positional Encoding
- Attention

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

- Learned or Fixed versions
- The paper uses fixed (as this can account for varying sentence length)
- Uses alternating Sin and Cos functions
- Y is the position of the word
- X is the length of word embedding (the paper uses 512)
- If dog was in a different position, it would have the same embedding value, this we need position to be somewhat accounted for.

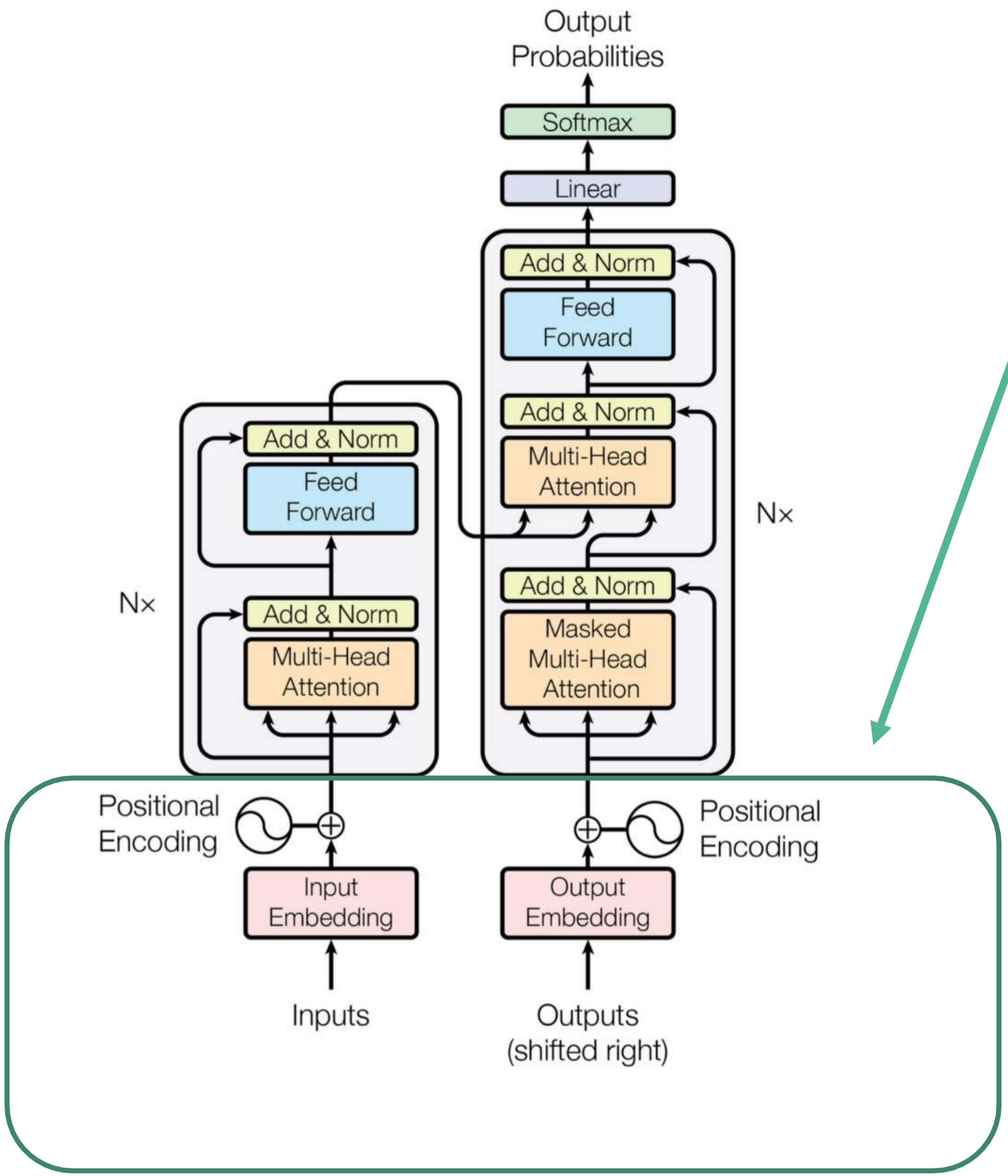


Dimensions

Word vectors

dog	-0.4	0.37	0.02	-0.34
cat	-0.15	-0.02	-0.23	-0.23
lion	0.19	-0.4	0.35	-0.48
tiger	-0.08	0.31	0.56	0.07
elephant	-0.04	-0.09	0.11	-0.06
cheetah	0.27	-0.28	-0.2	-0.43
monkey	-0.02	-0.67	-0.21	-0.48
rabbit	-0.04	-0.3	-0.18	-0.47
mouse	0.09	-0.46	-0.35	-0.24
rat	0.21	-0.48	-0.56	-0.37

animal
domesticated
pet
fluffy

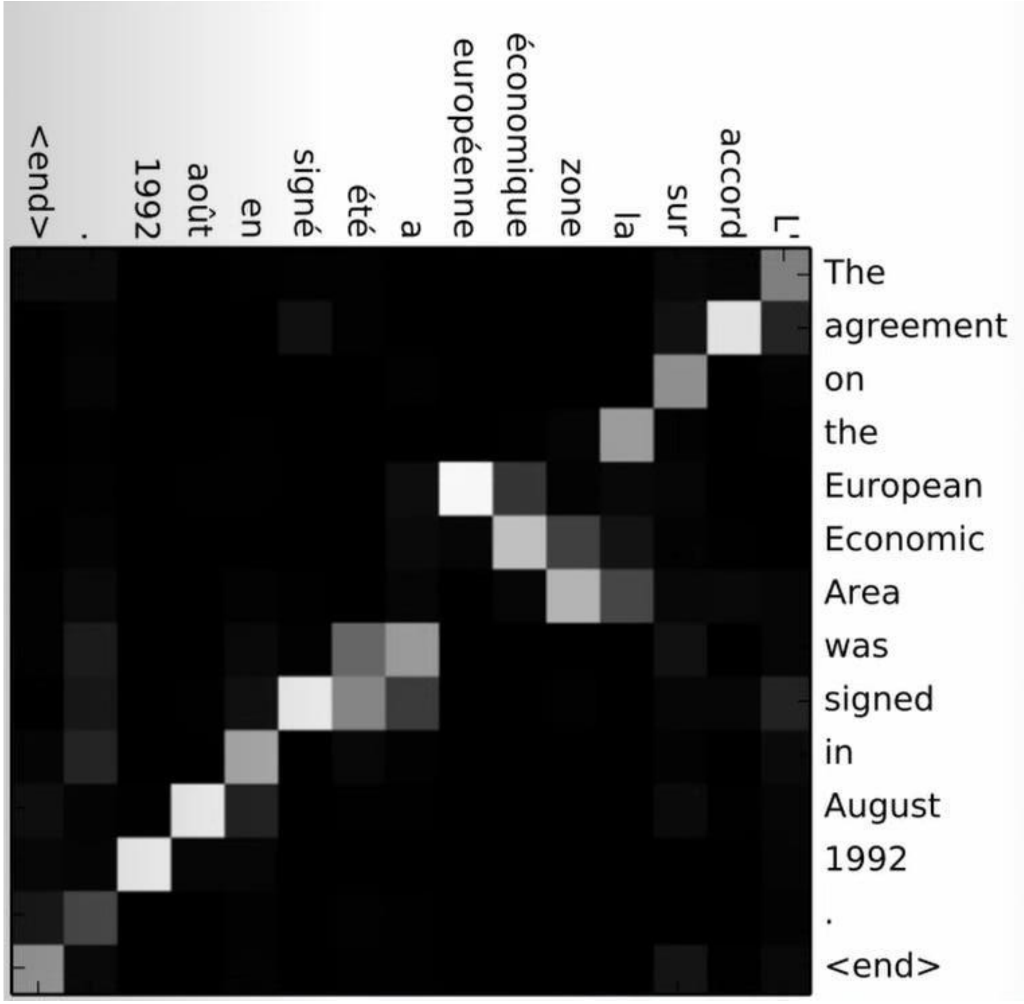


Prior to Attention

Prior to attention, the inputs are processed using embeddings, like in the LSTMs, this is dynamic.

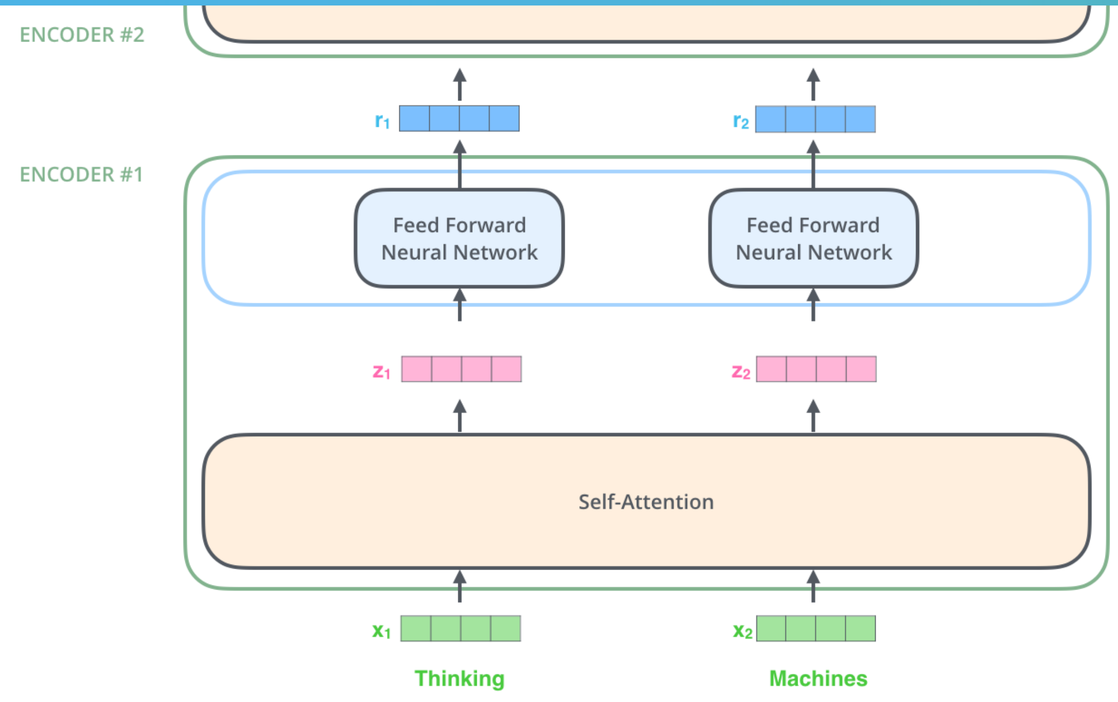
Original paper used a 512 dimension vector embedding.

Positional encoding is added, essentially the location of the word in a sentence.



Attention

- Attention is a method that does not rely on occurrence to identify the importance of a word in a sentence.
- The lighter heat map is the "importance" of the word in the sentence, the part we pay attention to.
- This same system can also work for images, like the heats maps from CNNs.



Attention (Self-Attention)

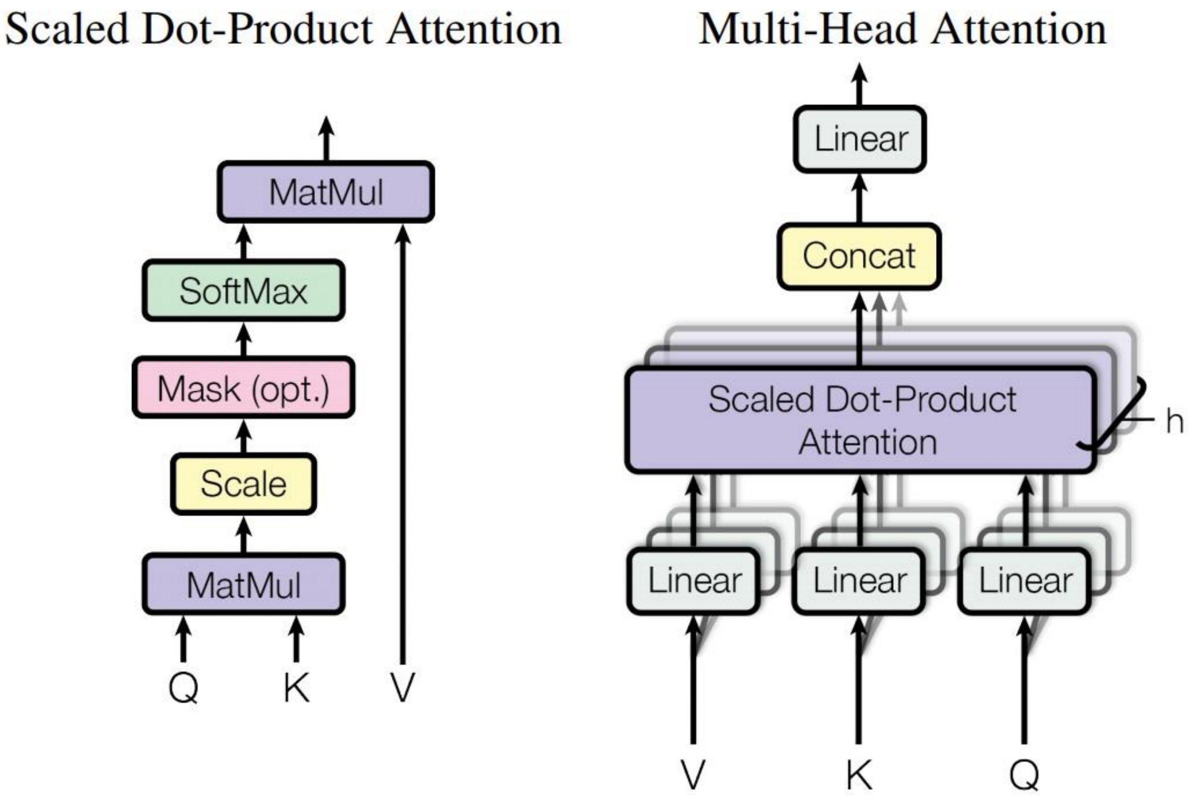
Sequence passed in (X1 and X2):

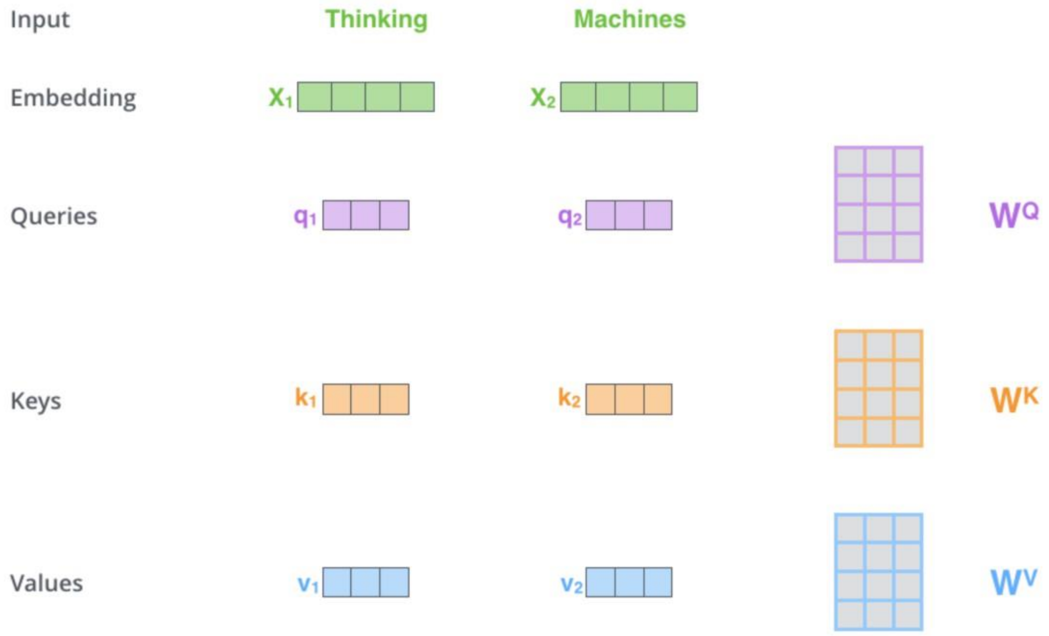
Let's briefly look at the internal components of an encoding/ decoding layer, before we go into them in detail:

- Attention is applied
  Keeping some relationships
- FFNN loses the association
  But allows parallel processing

Multi Head-Attention

Take self-attention and repeat multiple times to create multi-head attention

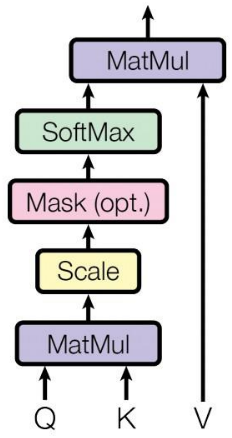


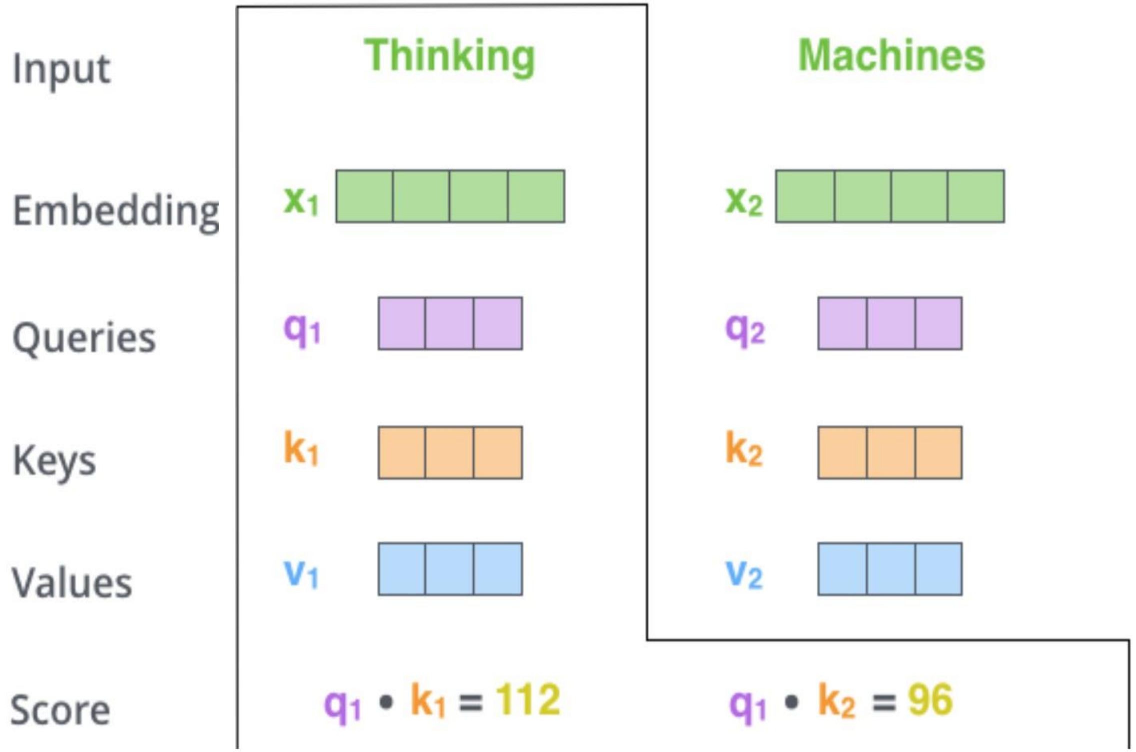


Attention

- Embedding has positional encoding.
- Multiply embeddings, by three matrices Query, Key and Value.
- These matrices are initialised randomly and updated during training (similar in a way to CNN kernel's).

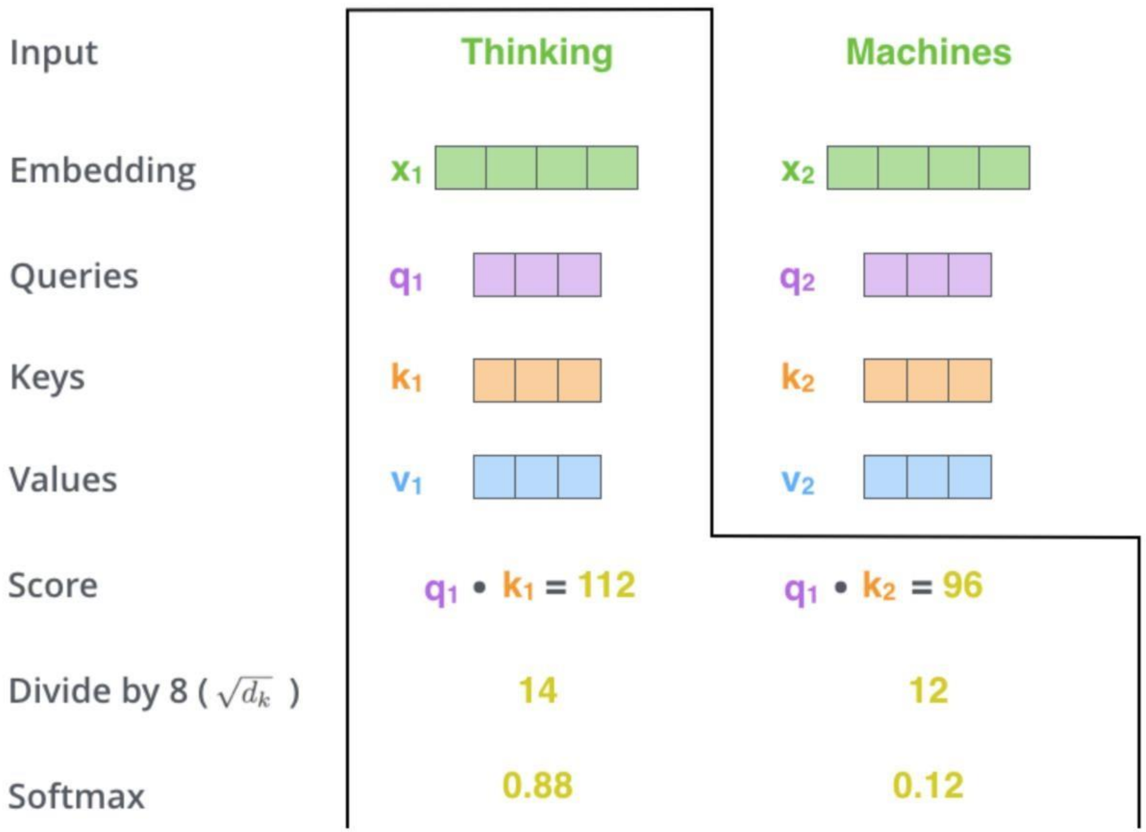
Scaled Dot-Product Attention





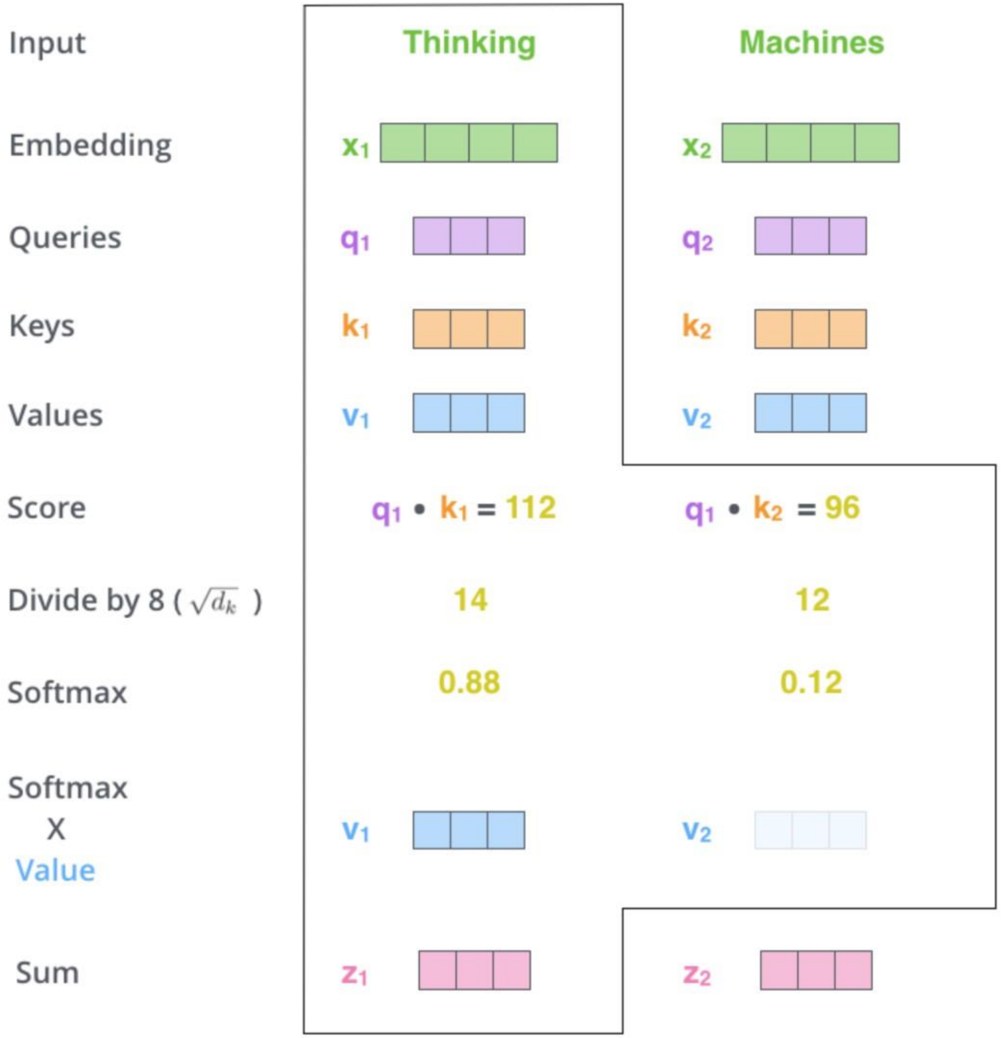
Attention – Calculate Scores

- Dot product of Query and key
- To get the Score of the first word vs second word we get query from first word and dot product this with key from second word.
- We now have score of first word vs all other words.
- Repeated for each word.
- Can be done in parallel.



Self-Attention

- Divide by 8
- Sounds random?
- Sqr 64 which is the length of the Q, K and V vectors.
- Passed through SoftMax to normalise values.



Self-Attention

SoftMax is generated for each word (related to the first word)

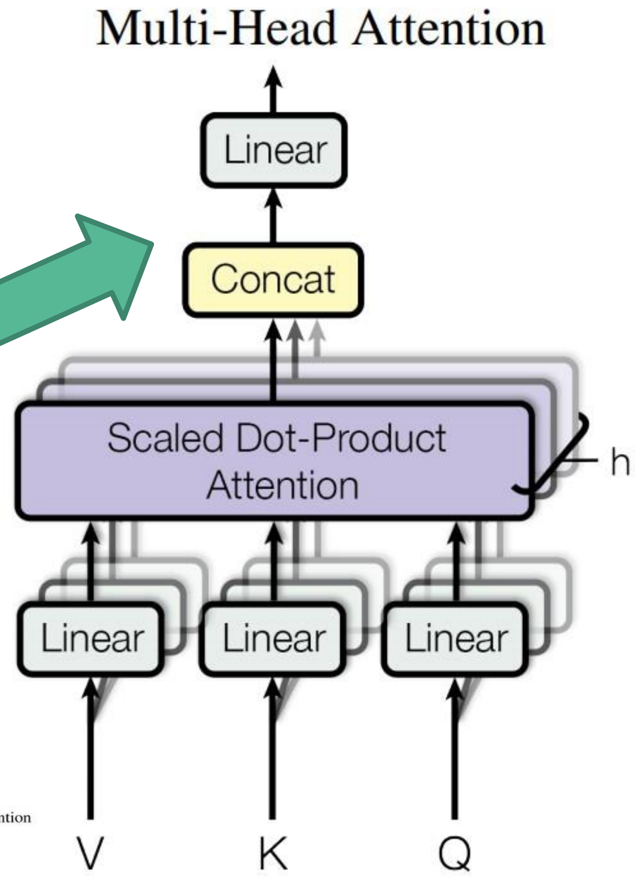
This is then multiplied the value for each layer.

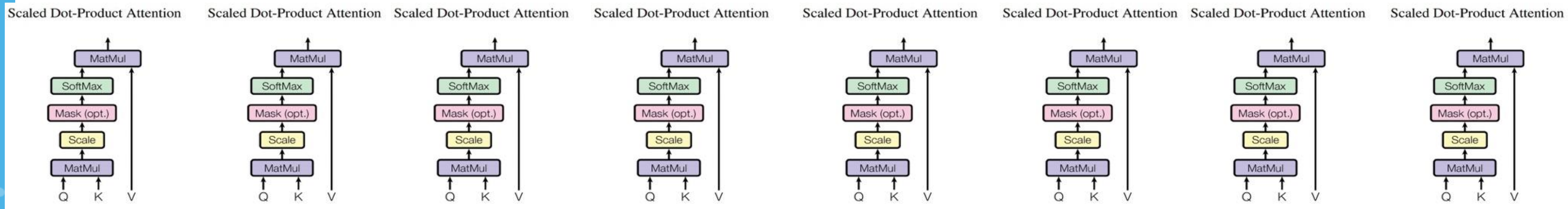
Finally, this is summed to generate the output for this word Z_1

Multi-Head Attention

Paper - 8 times

```python
embed_dim = 32  # Embedding size for each token
num_heads = 2  # Number of attention heads
ff_dim = 32  # Hidden layer size in feed forward network inside transformer

inputs = layers.Input(shape=(maxlen,))
embedding_layer = TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(20, activation="relu")(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(2, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

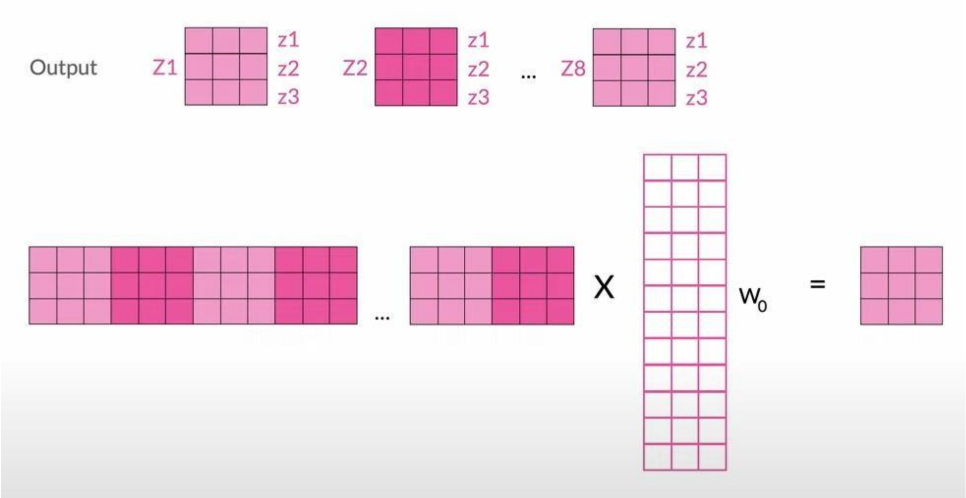


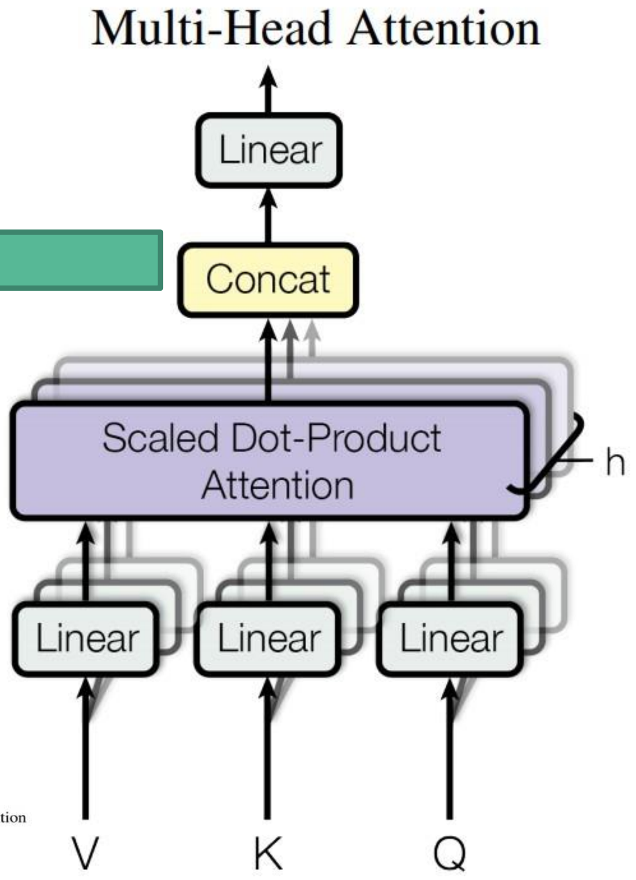


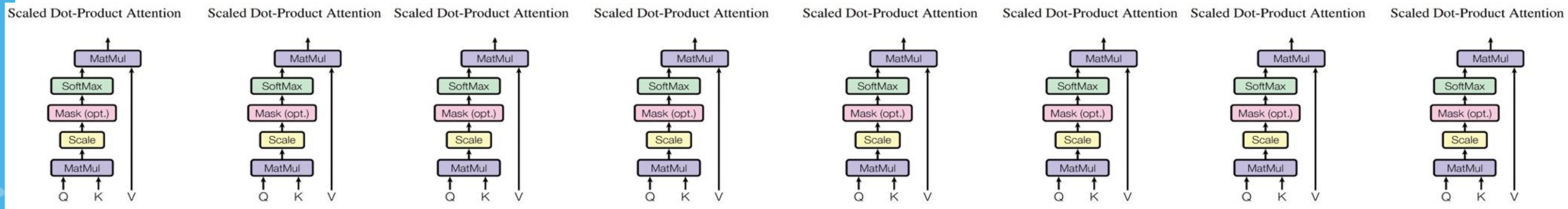
Multi-Head Attention

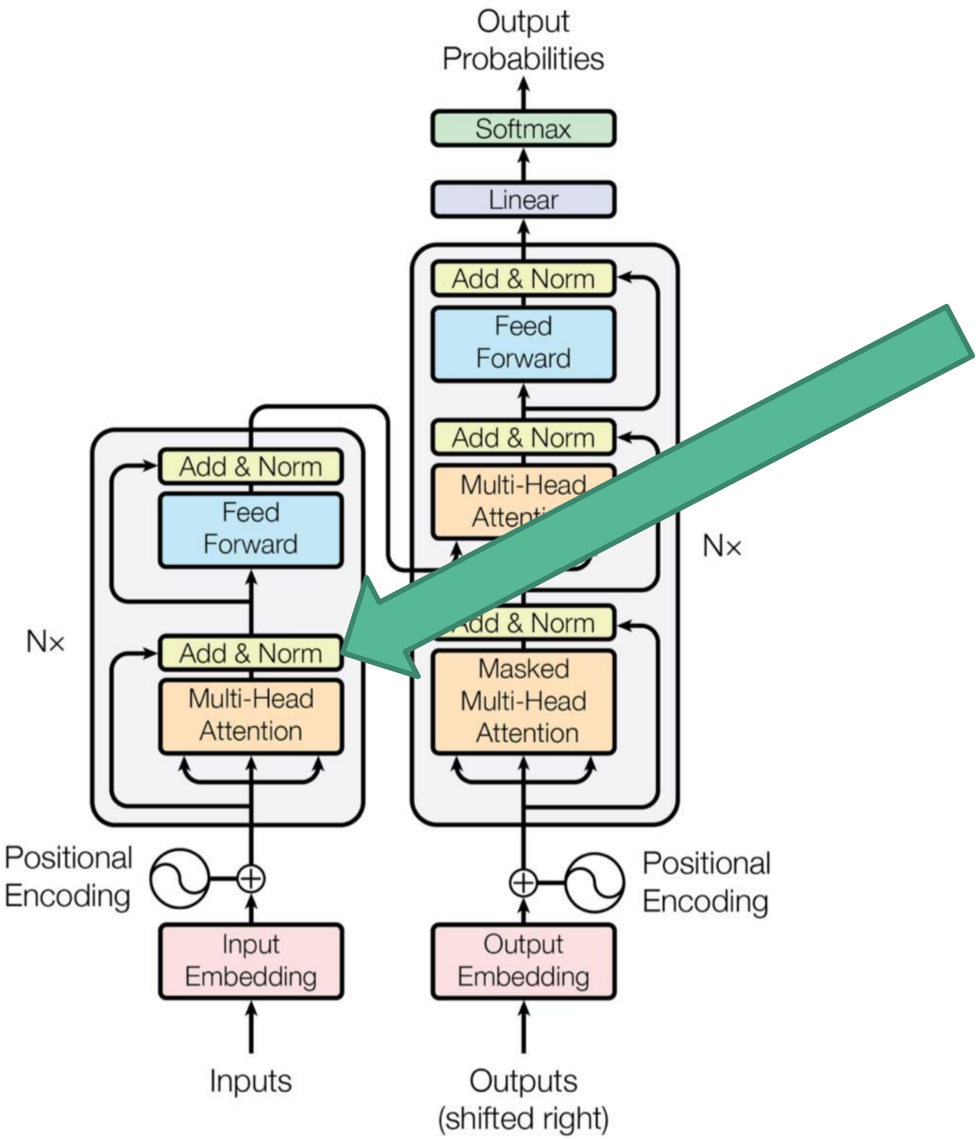
Paper - 8 times – this results in 8 Zn

To solve this we concatenate and multiple by an additional weight layer.







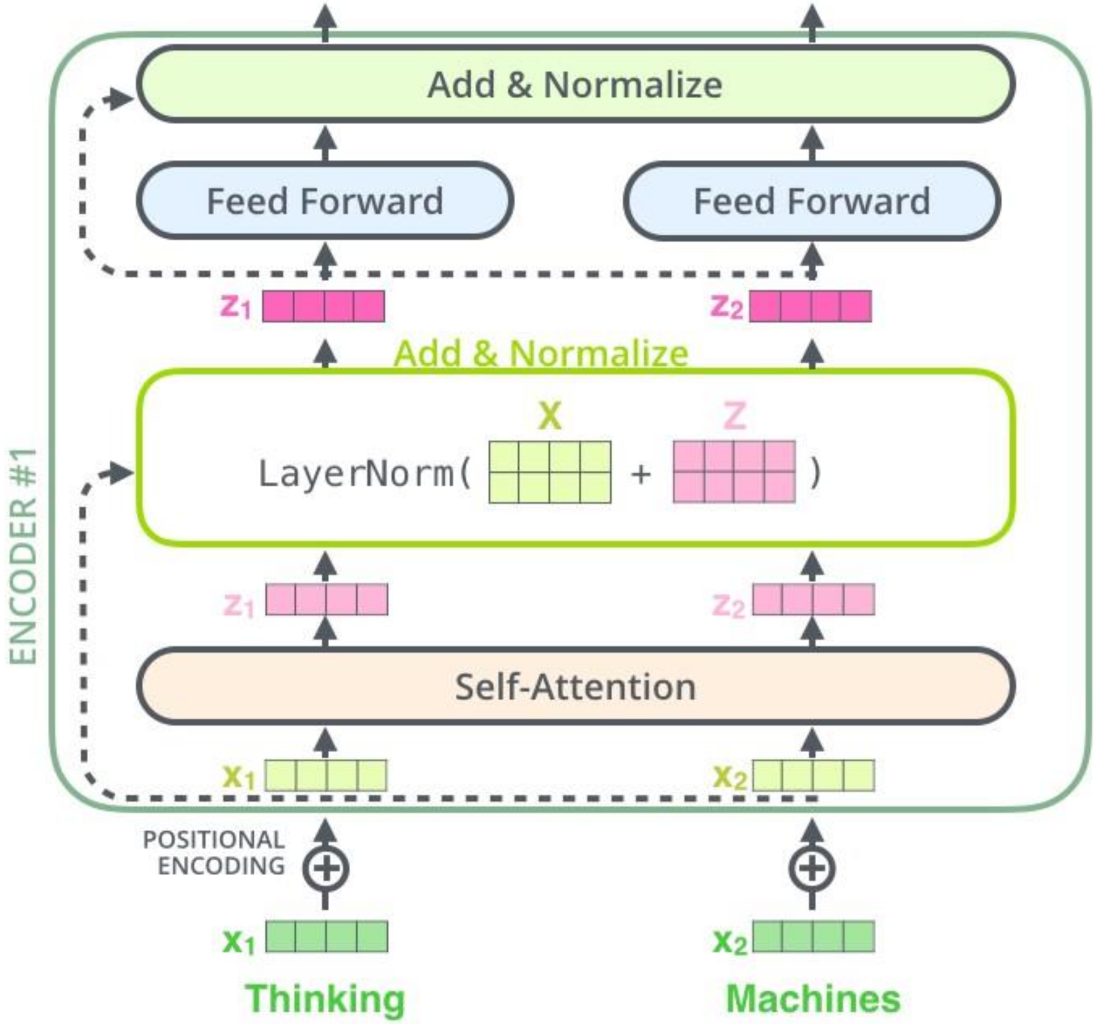


Normalization and Skip layers:

Improvements to the model:

- Data skips a layer (skip layers)
- Add and Norm

.

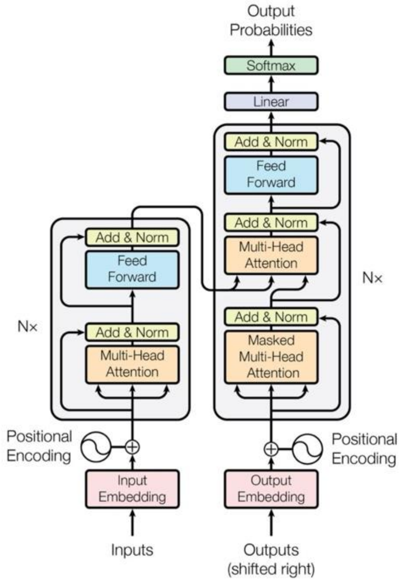


Normalization and Skip layers:

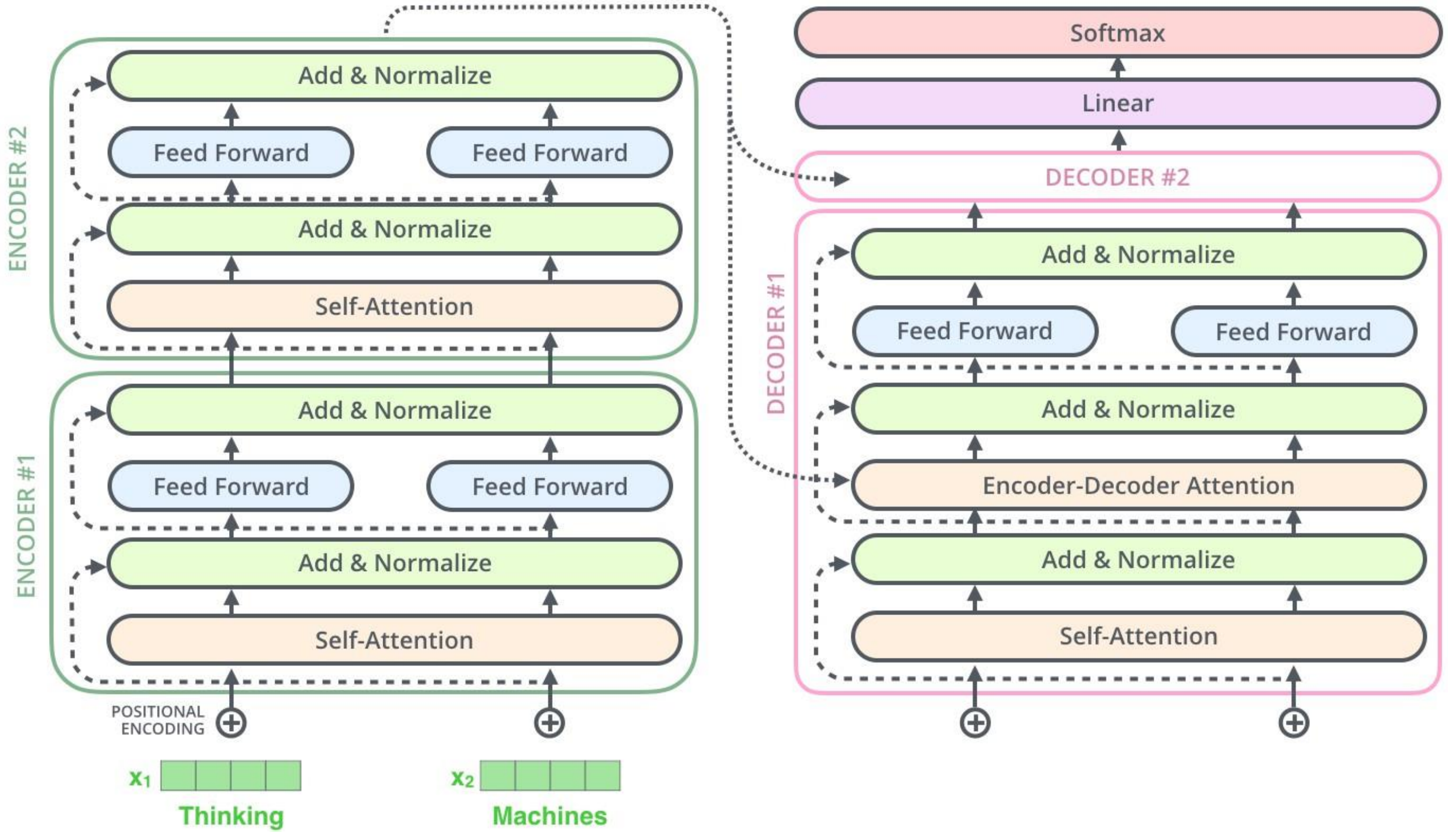
Improvements to the model:

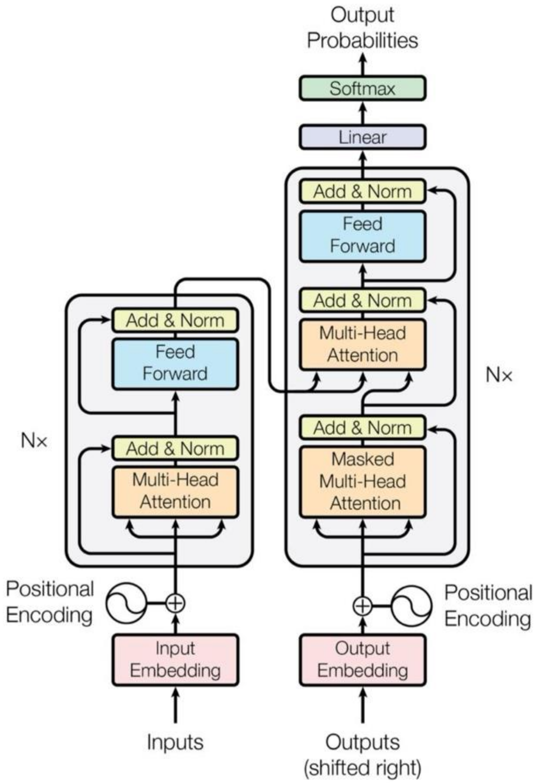
- Data skips a layer (skip layers)
- Add and Norm

.



Finally:


Softmax
Linear
DECODER #2
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
x1
Thinking
x2
Machines


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Thanks!

Zaur Gouliev

https://www.linkedin.com/in/zaurgouliev/

Email: zaur.gouliev@ucdconnect.ie